Foundations of Software Engineering

Lecture 8: Architectural Patterns, Tactics, and Evaluation

Claire Le Goues
Learning Goals

• Understand key parts of architectural process
• Use architectural styles and tactics for design decisions
• Make justified architectural decisions for new systems and within existing systems
• Review a proposed architecture
Traceability - Definition

"The ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and it's development process" – CoEST
Traceability in Requirements?
Traceability

Quality Goal: High Availability

Req.: Should run on redundant servers

OO Design/Impl.: Voting mechanism, socket communication

Architecture: Fault recovery with voting

Fault prevention: Regular restarts

Test: Killing random servers in test setup
Traceability Compliance

• Traceability required in some domains (avionics)
  – Why does X piece of code exist?
• "Enable verification of the absence of undocumented source code and verification of the complete implementation of the low-level requirements"
• Link to specifications and test procedures
Traceability and Architecture

• Architecture links quality attributes to the high-level and low-level system design
• Ensures quality attributes often not even visible in code
• Cost, effort, discipline needed to create and maintain.
  – Often incomplete, incorrect, outdated
• Developers hate it, and often do not understand the need.
  – "Unnecessary evil"
Case Study: The Google File System
Assumptions

• The system is built from many inexpensive commodity components that often fail.
• The system stores a modest number of large files.
• The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
• The workloads also have many large, sequential writes that append data to files.
• The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file.
• High sustained bandwidth is more important than low latency.

Qualities:
Scalability
Reliability
Performance
Cost

Figure 1: GFS Architecture

Questions

1. What are the most important quality attributes in the design?
2. How are those quality attributes realized in the design?
Exercise

For the Google File System, create a physical architecture view that addresses a relevant quality attribute.
So far in course

### Business Requirements Document

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Requirement Shopping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard based and interoperable messaging protocol</td>
<td>Messaging protocol must be based on industry standards to enable interoperability</td>
<td></td>
</tr>
<tr>
<td>Send Only</td>
<td>Also called Push MEP is simple one-way messaging where a message is sent with no expectation response.</td>
<td></td>
</tr>
<tr>
<td>Receive only</td>
<td>Also called Pull MEP is a message pattern where a non-addressable sender supports the ability to explicitly obtain messages from another application. This can be used for exchanges</td>
<td></td>
</tr>
<tr>
<td>Request/Response exchange</td>
<td>Message pattern consists of one or more request/response pairs. The correlation between request and a response is well defined. In this response maybe deferred and the requesting application may or may not block application processing until a response is received</td>
<td></td>
</tr>
<tr>
<td>Diagnostics</td>
<td>Authentication, diagnostic, logging &amp; routing information should be included in the message and not the payload</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Protocol capability to support assured and single delivery to the receiving application with no loss</td>
<td></td>
</tr>
</tbody>
</table>

Requirements

Implementation

Architecture
Levels of abstraction

- **Requirements**
  - high-level “what” needs to be done

- **Architecture** *(High-level design)*
  - high-level “how”, mid-level “what”

- **OO-Design** *(Low-level design, e.g. design patterns)*
  - mid-level “how”, low-level “what”

- **Code**
  - low-level “how”
What is architecture?

Architecture as structures and relations
(the actual system)

Architecture as documentation
(representations of the system)

Architecture as process
(activities around the other two)
Architectural Styles and Tactics
Architectural style (pattern)

• Broad principle of system organization
• Describes computational model
  — E.g., pipe and filter, call-return, publish-subscribe, layered, services
• Related to one of common view types
  — Static, dynamic, physical
Example Architectural Patterns

• System organization
  – Repository model
  – Client-server model
  – Layered model

• Modular decomposition
  – Object oriented
  – Function-oriented pipelining

• Control styles
  – Centralized control
  – Event-driven systems
Architectural style (pattern)

- Broad principle of system organization
- See reading

Source: codeproject.org
Architectural style (pattern)

Source: codeproject.org
Client-server pattern

• Separation of clients and servers
  – Servers provide services; known and “stable”
  – Clients request services; come and go

• Varieties: synchronous/asynchronous

• Impact on security, performance, scalability

• Examples: TCP, HTTP, X11
Where to validate user input?

Example: Yelp App

Example: There are a few times in life when a meal is so expertly crafted and planned that it is nothing short of genius. Last night, I had one of those meals - the Mahi Mahi.

The dish was excellently prepared. Grilled, juicy, and fresh without a hint of fishiness. A glaze of tangerine sauce brought a hint of tart sweetness. The fish was placed on a mound of sweet plantain rice. The combination of the fish and rice alone was too die for!
Client-server style
Layered system

Source: eclipse.org
Tiered architecture
Architectural Style?

Tactics

• Architectural techniques to achieve qualities
  – More tied to specific context and quality

• Smaller scope than architectural patterns
  – Problem solved by patterns: “How do I structure my (sub)system?”
  – Problem solved by tactics: “How do I get better at quality X?”

• Collection of common strategies and known solutions
  – Resemble OO design patterns
Many tactics out there!
Example Tactic Description: Record/playback

• Record/playback refers to both capturing information crossing an interface and using it as input into the test harness. The information crossing an interface during normal operation is saved in some repository and represents output from one component and input to another. Recording this information allows test input for one of the components to be generated and test output for later comparison to be saved.
Example Tactic Description: Built-in monitors

- The component can maintain state, performance load, capacity, security, or other information accessible through an interface. This interface can be a permanent interface of the component or it can be introduced temporarily via an instrumentation technique such as aspect-oriented programming or preprocessor macros. A common technique is to record events when monitoring states have been activated. Monitoring states can actually increase the testing effort since tests may have to be repeated with the monitoring turned off. Increased visibility into the activities of the component usually more than outweigh the cost of the additional testing.
Availability

Fault Detection
  - Ping/Echo
  - Heartbeat
  - Exception

Recovery-Preparation and Repair
  - Voting
  - Active Redundancy
  - Passive Redundancy
  - Spare

Recovery-Reintroduction
  - Shadow
  - State Resynchronization
  - Rollback

Prevention
  - Removal from Service
  - Transactions
  - Process Monitor

Fault Masked or Repair Made

Fault
Modifiability

- Localize Changes
  - Semantic Coherence
  - Anticipate Expected Changes
  - Generalize Module
  - Limit Possible Options
  - Abstract Common Services

- Prevention of Ripple Effect
  - Hide Information
  - Maintain Existing Interface
  - Restrict Communication Paths
  - Use an Intermediary

- Defer Binding Time
  - Runtime Registration
  - Configuration Files
  - Polymorphism
  - Component Replacement
  - Adherence to Defined Protocols

Changes Arrive → Changes Made, Tested, and Deployed Within Time and Budget
Performance

Resource Demand
- Increase Computation Efficiency
- Reduce Computational Overhead
- Manage Event Rate
- Control Frequency of Sampling

Resource Management
- Introduce Concurrency
- Maintain Multiple Copies
- Increase Available Resources

Resource Arbitration
- Scheduling Policy

Events Arrive → Response Generated Within Time Constraints
Security

Resisting Attacks
- Authenticate Users
- Authorize Users
- Maintain Data Confidentiality
- Maintain Integrity
- Limit Exposure
- Limit Access

Detecting Attacks
- Intrusion Detection

Recovering from an Attack
- Restoration
  - See Availability
- Identification
  - Audit Trail

System Detects, Resists, or Recovers from Attacks

Attack
Completion of an Increment →
  Manage Input/Output
    • Record/Playback
    • Separate Interface from Implementation
    • Specialized Access Routines/Interfaces
  Testability
    • Internal Monitoring
      • Built-in Monitors
  Faults Detected
Many tactics described in Chapter 5

Brief high-level descriptions (about 1 paragraph per tactic)

Second and more detailed third edition available as ebook through CMU library.
Architecture Design Process
What is architecture?

Architecture as structures and relations (the actual system)

Architecture as documentation (representations of the system)

Architecture as process (activities around the other two)
Architecture design process

• Choose part or whole system to focus on
• Understand relevant requirements
• Choose a notation
  – Type of view, vocabulary of elements
• Create a design
  – Patterns, tactics
• Evaluate
• Go vs no-go
  – Issues feed back into process
Choose scope \rightarrow Understand relevant requirements \rightarrow Choose a notation \rightarrow Create/Refine a design \rightarrow Evaluate

"Go"

Source: ACDM, ADD
Architectural decisions

• Heart of architecture – deciding which path to go
• Involve tradeoff analysis
• Representing the alternatives clearly – half of work
Architectural decisions

• Software architecture is *design*

  “Engineering design is [...] a decision-making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective” – ABET

• A decision is a step in the process
  – Record rationale! (not just diagrams)
  – Tradeoffs
  – Backtracking
Architecture design process

Choose scope ➔ Understand relevant requirements ➔ Choose a notation ➔ Create/refine a design ➔ Evaluate

Source: ACDM, ADD
Architectural decisions

• Software architecture is *design*

  “Engineering design is [...] a decision-making process (often iterative), in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective” – ABET

• A decision is a step in the process
  – Record rationale! (not just diagrams)
  – Tradeoffs
  – Backtracking
Architecture evaluation

• Goal: does the architecture satisfy requirements?
• ATAM – Architecture Tradeoff Analysis Method
  – Present requirements
  – Present architecture
  – Analyze architecture
  – Present results – risks and non-risks
Performance

Data Latency

Transaction Throughput

Modifiability

New product categories

Change COTS

Utility

Ability

H/W failure

COTS S/W failures

Security

Data confidentiality

Data integrity

(M,L) Minimize storage latency on customer DB to 200 ms.

(H,M) Deliver video in real time

(L,H) Add CORBA middleware in < 20 person-months

(H,L) Change web user interface in < 4 person weeks

(L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs

(M,M) Restart after disk failure in < 5 mins

(H,M) Network failure is detected and recovered in < 1.5 mins

(L,H) Credit card transactions are secure 99.999% of time

(L,H) Customer database authorization works 99.999% of time

Source: arnon.me
Athena – code review system

Source: Jansen and Bosch 2005
Source: Jansen and Bosch 2005
Architecture design process

Choose scope → Understand relevant requirements

Understand relevant requirements → Choose a notation

Choose a notation → Create/Refine a design

Evaluate → Create/Refine a design

Source: ACDM, ADD
Challenges of architecting

• Describe the system that is not built yet
• Domain knowledge is essential
• Huge space of options
• Heavily reliant on judgment
How much architecture?

• Design and document when needed, based on risk

• When:
  — Beginning
  — Whenever circumstances change

• Agile
How much architecture?

• YAGNI

• Risk

• When to start:
  – Before implementation
  – Circumstances change

• When to stop:
  – Well-defined, requirements addressed, passes evaluation
Source: Boehm and Turner 2003
Student application system

Source: Boehm and Turner 2003
Manned space mission software

Source: Boehm and Turner 2003
Challenges of architecting

• Describe the system that is not built yet
• Domain knowledge is essential
• Huge space of options
• Heavily reliant on judgment
Summary

Architecture as structures and relations
• Patterns
• Tactics

Architecture as documentation
• Views
• Rationale

Architecture as process
• Decisions
• Evaluation
• Reconstruction
• Agile
References

Further Readings

Levels of abstraction

• Requirements
  – high-level “what” needs to be done

Architecture
  (High-level design)
  – high-level “how”, mid-level “what”

OO-Design
  (Low-level design, e.g. design patterns)
  – mid-level “how”, low-level “what”

• Code
  – low-level “how”
Architecture reconstruction

• Goal: describe architecture of an existing system given its source code
• Difficulty: level of abstraction in programming language is too low
• Process:
  – Iterative
  – Interpretive
  – Interactive
Reconstruction steps

• Extract *raw views*
  – Tool assistance, static & dynamic analysis

• Construct a database
  – Aggregate large volumes of data

• View fusion
  – Synthesize a hypothetical view

• Check for violations
Architecture reconstruction

- Iterative
- Interpretive
- Interactive
Reconstruction steps

- Extract *raw views*
  - Tool assistance, static & dynamic analysis
- Construct a database
  - Aggregate large volumes of data
- View fusion
  - Synthesize a hypothetical view
- Check for violations