Architecture Haiku

George Fairbanks

3 August 2010

Rhino Research

Software Architecture Consulting and Training

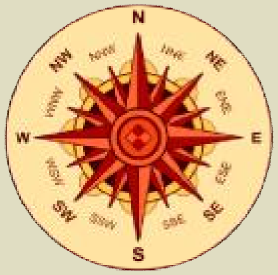
http://RhinoResearch.com

Talk overview

- Architecture descriptions tend to be verbose
  - E.g.: Documentation package
  - Complete, rather than suggestive
- Problem: We have caviar taste; McDonald’s budget
- Q: What if we only use one page?
  - Must use concise language
  - High power-to-weight ideas
- A: Architecture Haiku
  - Tradeoffs, quality attribute priorities, drivers, design rationales, constraints, architectural styles
- Future
  - Requires shared knowledge: terms, styles
  - Will Haiku help architecture education?

Talk outline

- Introduction
  - About me, software architecture, documentation packages, agile, forces, concision
- Haiku How-To
- Haiku Example: Apache
- Group exercise
- Future



What is software architecture?

The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both. [Clements et al., DSA2, 2010]

- **In loose language:**
  - **It's the macroscopic organization of the system**

- **Must keep these ideas separate:**
  - X **The job title/role "architect"**
  - X **The process of architecting/designing (also: when)**
  - ü **The engineering artifact called the architecture**

- **Every system has an architecture**
  - **Identify it by looking back (avoids tangling with process & roles)**
  - **E.g., "Aha, I see it is a 3-tier architecture"**

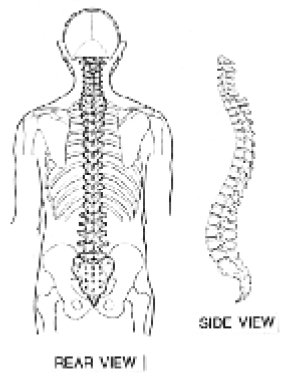
Architecture as skeleton

- An animal's skeleton:
  - Provides its overall structure
  - Influences what it can do

- Examples
  - Birds fly better because of wings and light bones
  - Kangaroos jump because of leg structure
  - Convergent evolution: Bat skeletons have wings

- Tradeoffs
  - 4 legs faster vs. 2 legs easier to use tools

- Software skeleton examples
  - 3-tier: localize changes, concurrent transactions
  - Cooperating processes: isolate faults
  - Peer-to-peer: no central point of failure



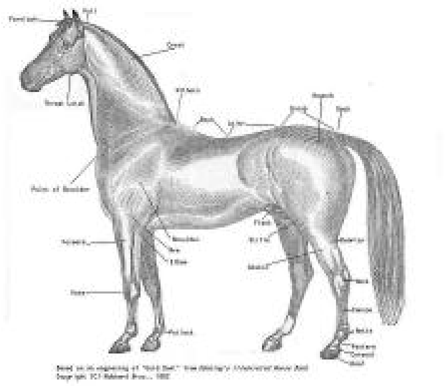
Architecture influences quality attributes

- Function: Carrying apples to market
  - Solution 1: Use humans
  - Solution 2: Use horses

- Both solutions work
  - Yet differ in their quality attributes

- Quality attributes
  - A.k.a. extra-functional requirements, the "ities"
  - E.g., latency, modifiability, usability, testability

- Architecture influences the system's quality attributes
  - E.g., pipe and filter is reconfigurable
  - E.g., map-reduce is scalable

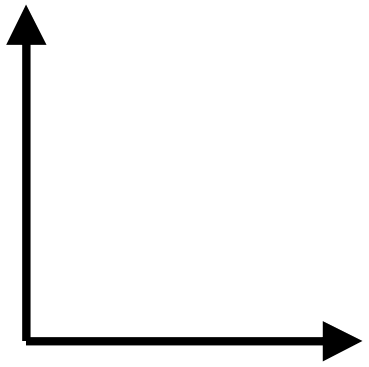


Architecture orthogonal to functionality

- Architecture orthogonal to functionality
  - Can mix-and-match architecture and functionality
  - Can (mostly) build any system with any architecture

- Shift in thinking
  - No: good or bad architectures
  - Yes: suitable or unsuitable to supporting the functions

- Architecture can help or hurt you achieve functionality
  - Help example: use horses to transport apples
  - Hurt example: use horses to stack apples



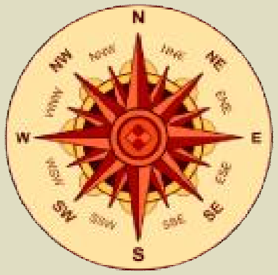
Documentation packages

- What is a documentation package?
  - Complete architecture description
  - See Documenting Software Architectures, 2nd Ed, 2010

- But:
  - Hard to reconcile with Agile processes
  - Expensive
  - Not terse examples

Talk outline

- Introduction
- Haiku How-To
  - Tradeoffs, quality attribute priorities, drivers, design decisions, constraints, architectural styles
- Haiku Example: Apache
- Group exercise
- Future



Idea: Architecture Haiku

- Goal: Best 1-page architecture description

- What to include?
  - High power-to-weight items
  - Items that promote insight, not comprehensiveness

- Techniques
  - Concise language (e.g., technical terms)
  - Document differences
  - Hints at critical junctions

- Implications (i.e., who can read the haiku?)
  - Need shared technical terms
  - Need shared conceptual model

Haiku Contents

1. Solution description
2. Tradeoffs
3. Quality attribute priorities
4. Architecture drivers (QA scenarios)
5. Design rationales
6. Constraints
7. Architecture styles
8. Diagrams

1. Solution description

- Simple text describing the system

- Good to include:
  - Most important functions
  - Goals

- Challenges
  - Verbosity

- Examples
  - XBMC is a cross-platform music and video player that supports every major media format.
  - LyX is a structured document processor, suitable for technical writing, that uses LaTeX to render professional-quality PDFs.

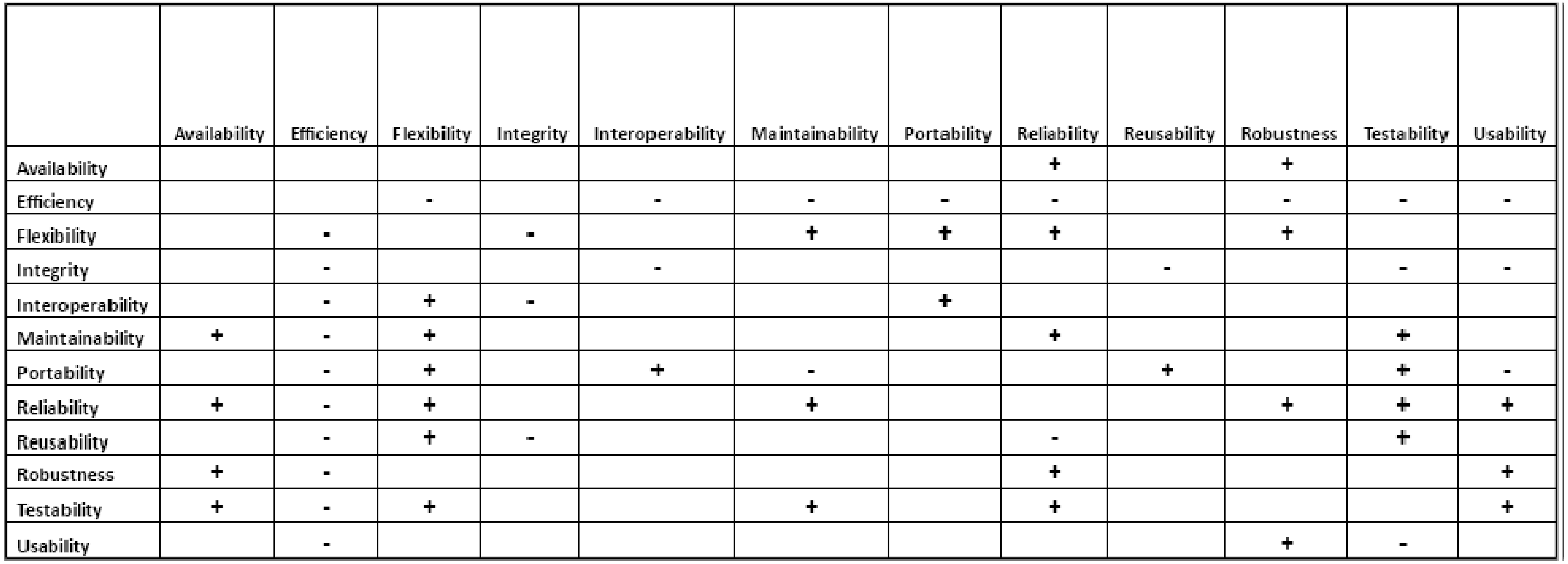
2. Tradeoffs

- Tradeoff: More of this à less of that
- Examples
  - Portability vs. playback efficiency. Platform-specific resources (e.g., dedicated hardware) often provide media playback benefits, including efficiency, yet using these resources ties the software to that platform
  - Weight vs. speed. The heavier a car is, the slower it accelerates.
- Everything trades off against cost

QA tradeoffs

- **Domain tradeoffs or system-specific tradeoffs**
  - **Arise from domain quirk, or particular design**

- **Generic quality attribute tradeoffs**
  - **E.g., generally *efficiency* trades off against *maintainability***



	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability								+		+		
Efficiency			-		-	-	-	-		-	-	-
Flexibility		-		-		+	+	+		+		
Integrity		-			-				-		-	-
Interoperability		-	+	-			+					
Maintainability	+	-	+					+			+	
Portability		-	+		+	-			+		+	-
Reliability	+	-	+			+				+	+	+
Reusability		-	+	-				-			+	
Robustness	+	-						+				+
Testability	+	-	+			+		+				+
Usability		-								+	-	

Table from Wiegers, Software Requirements, 2003, via Morgan "Implementing System Quality Attributes", 2007.

3. Quality attribute priorities

- Goal: Prioritized list of quality attributes
    - Some QA's likely not relevant, or very low priority
    - Some QA's critical to project success

- For credit card processing:
    - Security > latency > throughput

- For an MP3 player:
    - Usability > audio fidelity > extensibility

- Make an argument for different prioritizations
- What factors influence the prioritization?

4. QA scenario structure

<u>QA Scenario Templates</u>

- **Basic template**: stimulus and response
    - Stimulus: agent or situation that triggers scenario
    - Response: reaction to stimulus

- **Ideal template**: Add source, environment, response measure
    - Stimulus: as above
    - Response: as above
    - Source: Who/what creates stimulus
    - Environment: mode of the system. E.g., normal or low demand.
    - **Response measure**: testable response (e.g., “happens in 2ms”)

<u>QA Scenario Examples</u>

- **Basic scenario**: System allows rapid scanning of book copies.

- **Ideal scenario**: Under normal conditions, when a librarian scans a book copy for checkout, the system updates its records and is ready to scan the next one within 0.25 seconds.

QA scenarios from Bass et al., Software Architecture in Practice, 2003

Architecture drivers

<u>**Architecture Drivers**</u>

- Each QA scenario can be graded by:
  - Importance to stakeholder (high, medium, low)
  - Difficulty to implement (high, medium, low)
- Architecture drivers are
  - QA scenarios
  - or functional scenarios (eg use cases)
  - that are rated (H,H)

<u>**Examples**</u>

- S1 (H,H): When a librarian scans a book copy for checkout, the system updates its records and is ready to scan the next one within 0.25 seconds.
- S2 (M,H): When librarian station cannot contact the main system, librarians can continue to check books in and out.
- S3 (H,M): The system can be modified to use a different source of borrower entitlements within 1 programmer week of effort.

5. Design rationales

- Design rationales explain why
- They should align with your quality attribute priorities

  <x> is a priority, so we chose design <y>, accepting downside <z>.

- An example:
  - Since avoiding vendor lock-in is a high priority, we choose to use a standard industry framework with multiple vendor implementations, even though using vendor-specific extensions would give us greater performance.
- But: Good intentions can go awry
  - E.g., performance optimization hindering modifiability

6. Constraints (Guiderails)

- Developers voluntarily constrain systems
  - Counter-intuitive
  - Ensures what a system does not do
  - I.e., guiderails
- Constraints help ensure outcomes
  - E.g., ensure quality attributes are met
  - No constraints = no analysis
- Examples
  - Plugins must use cross-platform API to read files à portability
  - EJBeans must not start own threads à manageability
  - EJBeans must not write local files à distribution

7. Architectural styles

- Examples
  - Big ball of mud
  - Client-server
  - Pipe-and-filter
  - Map-reduce
  - N-tier
  - Layered
  - ...

- Each predefines
  - Elements (e.g., pipes, map functions)
  - Constraints, ...

- Benefits
  - Known tradeoffs
  - Known suitability
  - Compact term for communication

See more architectural styles in my book (of course) or wikipedia

8. Diagrams

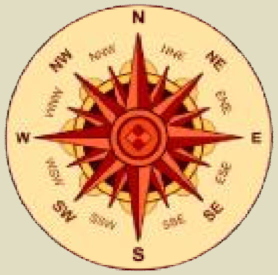
Viewtype	Contents
Module	Modules, Dependencies, Layers
Runtime	Components, Connectors, Ports
Allocation	Servers, Communication channels

- Three primary viewtypes: Module, Runtime, Allocation
  - Many views within a viewtype
  - View suitability

- Challenge
  - Remember: Just one page!
  - Small diagrams only

Talk outline

- Introduction
- Haiku How-To
- Haiku Example: Apache
- Group exercise
- Future



The Apache Web Server

- History
  - Evolved from UIUC NCSA server

- Users
  - From Hosting providers to mom-and-pop

- Notable characteristics
  - Cross-platform (via Apache Portable Runtime layer)
  - Extensible (via pluggable modules)
  - Configurable (via text files)
  - Interoperable (e.g., with app servers)

Apache as a Haiku

1. Description
   - The Apache HTTP Server serves web pages/requests, is extensible by third parties, and integrates with procedural code (e.g. CGI scripts, app servers)

2. Tradeoffs
   - Textual over GUI config: ssh access; scriptability
   - Configurability over usability: for professional sysadmins; expert over casual use
   - Extensibility over performance: distributed OSS creation of new modules

3. Top 3 quality attributes, prioritized
   - Extensibility > Configurability > Performance

4. Architecture drivers
   - Third party writes a new extension module in 1 week
   - ISP configures new virtual host via script
   - Server ported to new operating system in 1 month

Apache as a Haiku

5. Design rationales
   - Since extensibility is more important than performance, modules are dynamically configured to process requests, potentially increasing latency

6. Constraints (guide rails)
   - All OS calls must go through Apache Portable Runtime layer

7. Architectural styles
   - Client-server: Browsers to main web server
   - Pipe-and-filter: Requests and responses processed through network of filter modules (e.g., URL rewrite, compress)

8. Diagrams
   - Next pages

Client-server diagram

- **Apache, with clients and administration**
- **Runtime viewtype**

- **Diagram not surprising enough to include**

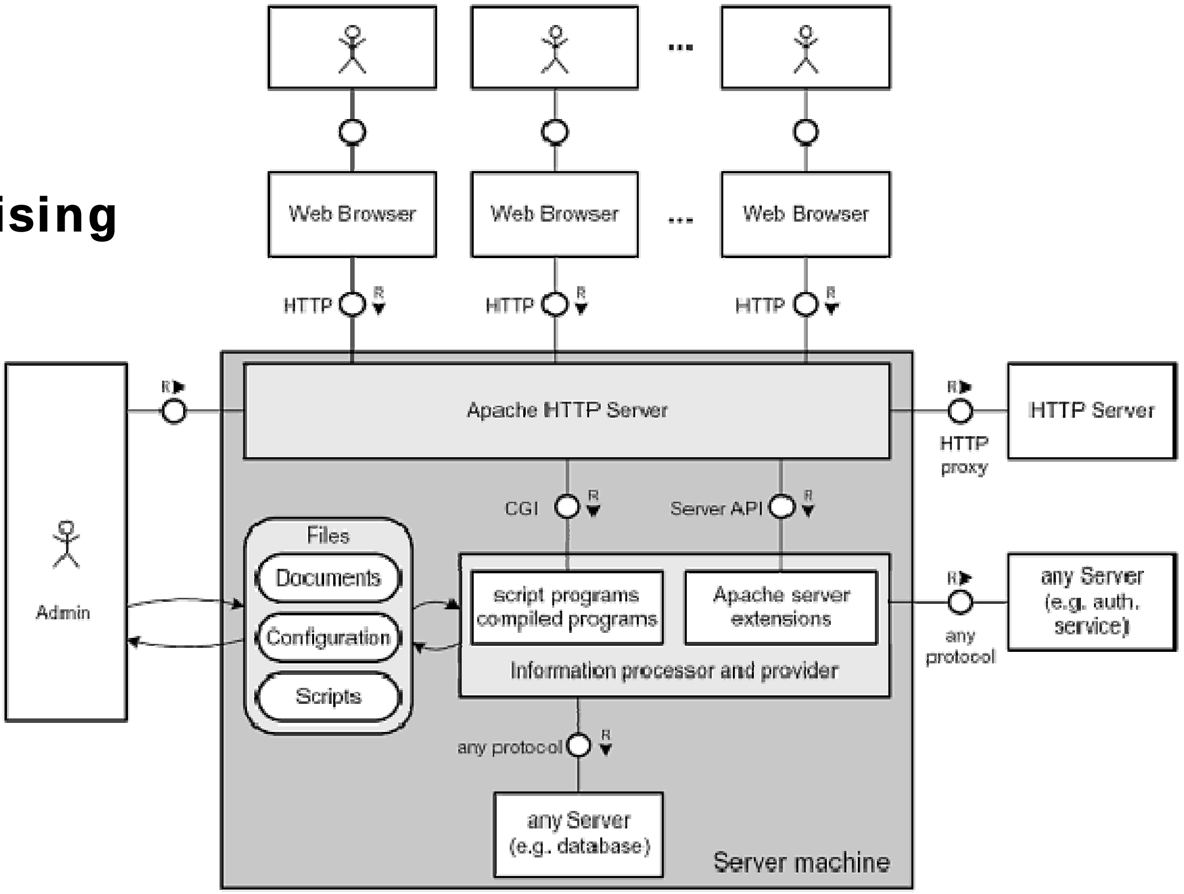


Diagram from: Apache Modeling Project:
Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel, Oliver Schmidt
http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html

Pipe-and-filter diagram

- **Request processing**
- **Apache (dynamically) processes requests**
  - **Input pipeline**
  - **Output pipeline**

- **Note: Runtime viewtype**

- **Perhaps worthy to include**

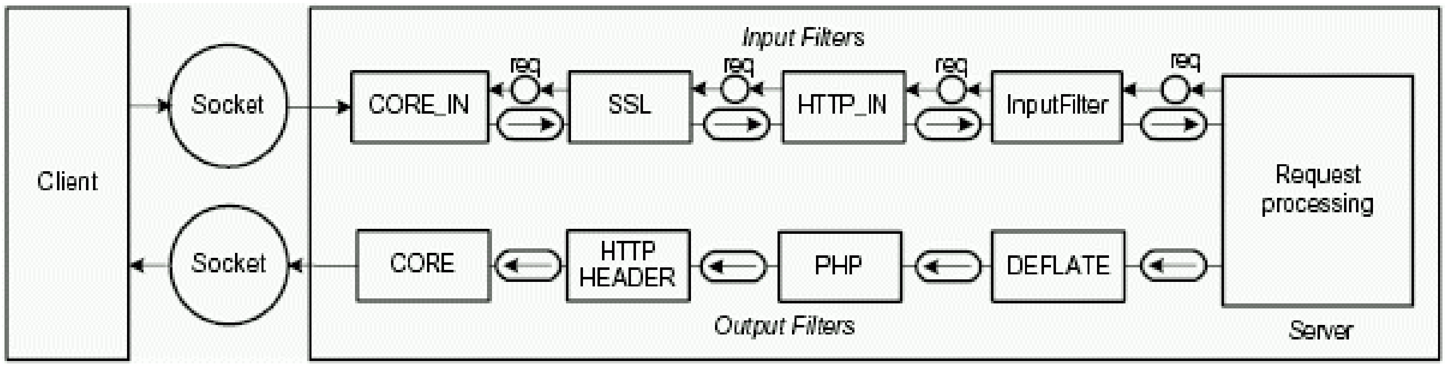
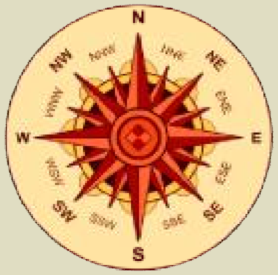


Diagram from: Apache Modeling Project:
Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel, Oliver Schmidt
http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html

Talk outline

- Introduction
- Haiku How-To
- Haiku Example: Apache
- Group exercise
    - Airport screening
- Future



Airport screening exercise

- Choose your descriptions from this list:

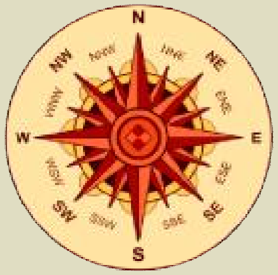
1. Solution description
2. Tradeoffs
3. Quality attribute priorities
4. Architecture drivers (QA scenarios)
5. Design rationales
6. Constraints (guide rails)
7. Architecture styles
8. Diagrams

- Tips

- Incomplete descriptions
- Suggestive, not comprehensive
- Hints at critical junctions

Talk outline

- Introduction
- Haiku How-To
- Haiku Example: Apache
- Group exercise
- Future
    - Role of architecture, terminology, learn from (Haiku) examples,



Implications & future

- Role of architecture
  - Acts as skeleton
  - Makes QA’s easy or hard to achieve
  - Today, often ignored

- Assumed architectural knowledge
  - Terminology (style names, components, connectors, …)
  - Deltas

- Learning from (Haiku) examples
  - Imagine a book of Haiku examples
  - OSS rarely documents architecture
  - Let’s build a catalog (email me)

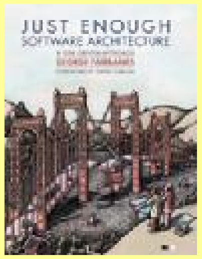
- Candidate agile technical practice

Conclusion

- 3 distinct ideas:
  - The job title/role “architect”
  - The process of architecting/designing
  - The engineering artifact called the architecture

- Architecture Haiku: 1-page description
  - Incomplete, document deltas, hints at critical junctions
  - High power-to-weight items that yield insight

- Contents
  - Tradeoffs, QA priorities, drivers, rationales, constraints, styles, and maybe diagrams



Parts of this talk are excerpted from my book
Just Enough Software Architecture: A Risk-Driven Approach
http://RhinoResearch.com

Shameless plugging

- E-book available now $25
  - One price, 3 formats
  - PDF, ePub, Mobi
  - No DRM
  - http://RhinoResearch.com
- Hardback in ~5 weeks
  - Amazon.com
  - About $40 street price