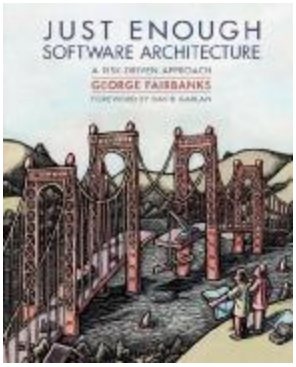
Architecture Haiku

George Fairbanks

24 June 2011



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**
- Conceptual models
- Architecture haiku
- Haiku example: Apache web server
- Group exercise
- Discussion

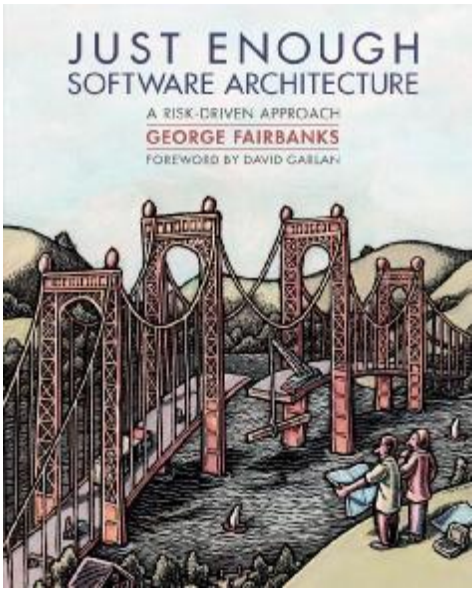
How I normally give this talk vs today

- Normally:
  - To:
    - developers / Agilists
  - Short-term goals:
    - make architecture less scary
    - explain some of our thinking
    - show value
  - Long-term goal:
    - bridge Agile and Architecture communities

- Today
  - To:
    - architecture experts! (No explaining needed)
  - Short-term goals:
    - share and workshop an idea
    - 1-page architecture descriptions
  - Long-term goal:
    - Same

About me (George Fairbanks)

- PhD Software Engineering, Carnegie Mellon University
- Thesis on frameworks and static analysis (Garlan & Scherlis advisors)
- Program committee member: WICSA 2009, ECSA 2010, ICSM 1009; CompArch 2011 local chair
- Architecture and design work at big financial companies, Nortel, Time Warner, others
- Teacher of software architecture, design, OO analysis, EJB
- Author: *Just Enough Software Architecture*



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]

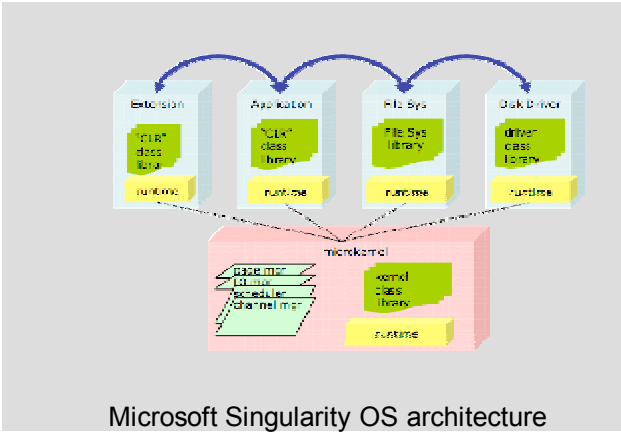
- In loose language:
  - It's the macroscopic organization of the system
- Must keep these ideas separate:
  - The **job title/role** "architect"
  - The **process** of architecting/designing (also: when)
  - The **engineering artifact** called the architecture
- Every system has an architecture
  - Identify it by looking back (avoids tangling with process & roles)
  - E.g., "Aha, I see it is a 3-tier architecture"
- Vary in **scale** and **complexity**

Low complexity & large scale

- Large scale partitioning
- Systematic interconnections
- Repeated patterns

} **Comprehensibility**

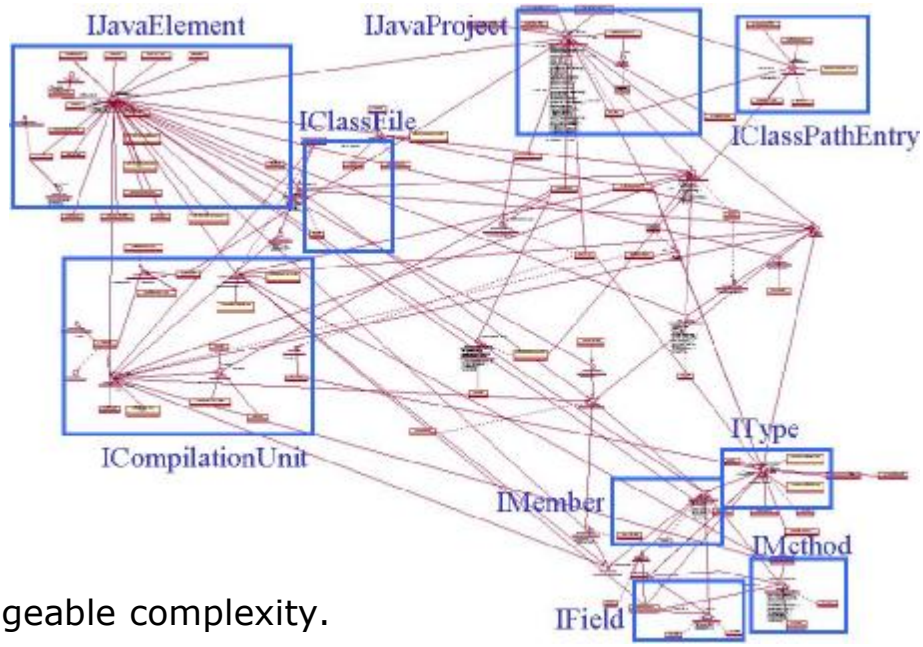
- Comprehensibility: no accident
  - Want to verify the OS
  - Complexity → no verification
- Low complexity
  - Must be chosen
  - Unnatural
  - Not emergent
- Your brain
  - Perhaps very large
  - Yet, only so big



Microsoft Singularity OS architecture

High complexity & small scale

- Q: Are the blue boxes encapsulation boundaries?
  - If **yes**: can treat them as black boxes
  - If **no**: must understand their insides



- Let's call this manageable complexity.
  - ~ 100 classes
  - Cannot keep going indefinitely
  - If does, Big Ball of Mud

Diagram from Grady Booch, presentation on Software Archaeology

Scale and complexity: summary

- Problem:
  - Scale and complexity are increasing
  - Hard to tolerate both
- Solution 1:
  - Avoid building large, complex systems
- Solution 2:
  - Reduce complexity
  - … but how?

	Low Scale	High Scale
Low Complexity	**Yay!** (Doghouses)	**OK**
High Complexity	**OK**	**Danger!** (Mars)

Idea: Match architecture detail to project

- Projects vary in size and complexity
- Dial of architecture rigor / detail
  - doghouse vs battleship
- ATAM vs Haiku – no contest, or horses-for-courses?
- Goldilocks: just right

The Risk-Driven Model

- **The Risk-Driven Model:**
  1. Identify and prioritize risks
  2. Apply relevant architecture activities
  3. Re-evaluate

- **Must balance**
  - Wasting time on low-impact techniques
  - Ignoring project-threatening risks

- **Q: On Agile projects, which architecture activities should/can you apply?**

Ron ArmsCtrong, CC

What can architects do to help developers?

Regular developers are turning to Agile

- CompArch/WICSA 2011 attendance: ~150
- Agile 2010 attendance: ~1400

- Good developers are Agilists
  - They listen to Kent Beck instead of Len Bass
  - Danger: Our message is not heard

How Agilists design: Agile design techniques

- Architectural Spikes / Spike Solutions
- Domain Driven Design
- Emergent Design / Evolutionary Design
- CRC Cards
- Design by Contract
- System Metaphor
- Refactoring

What can architects contribute to Agile?

- Q: What is our **message** to Agilists? That is, what is it that we want to teach them to do better?
  - A1: Process/roles: BDUF, corner office architects, waterfall
  - A2: Paper: Documents, documents, documents
  - A3: Equations: Formalisms
  - A4: Conceptual model: How architects look at problems and solve them – a way of thinking and perceiving the world
  - A5: Checklists (to get at QAs)

Then, what's the vehicle?

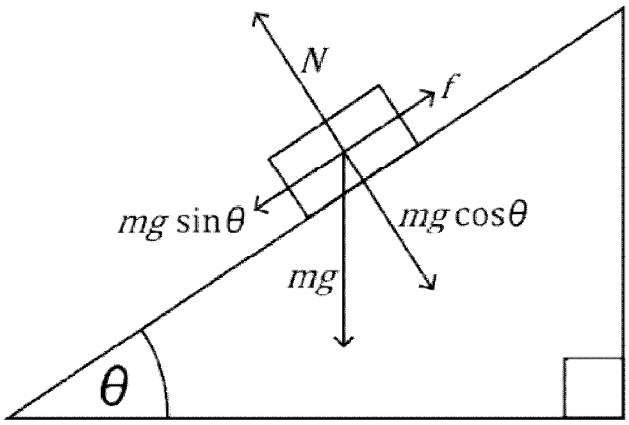
- Assume we agree on what architects have to teach Agilists, the next question is:
- Q: Which of our techniques should transmit our message to Agilists?
  - A1. We reuse as-is any architecture techniques
  - A2: We invent new Agile-compatible techniques
- My opinion:
  - We should transmit our **conceptual model of architecture** to Agilists using **new Agile-compatible techniques**

Talk outline

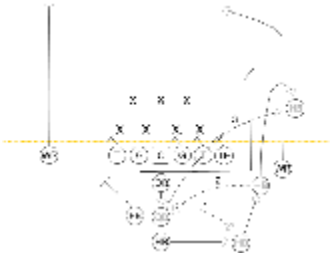
- Introduction
- **Conceptual models**
- Architecture haiku
- Haiku example: Apache web server
- Group exercise
- Discussion

What is a conceptual model?

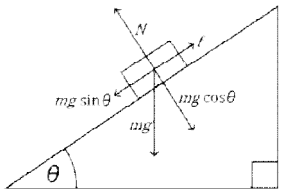
- What is a conceptual model?
  - A **conceptual model** is a set of concepts that can be imposed on raw events to provide meaning and structure.
- It organizes chaos
  - Enables intellectual understanding
  - Fits big problems into our finite minds
- Synonyms:
  - Conceptual framework
  - Mental model



Examples of conceptual models

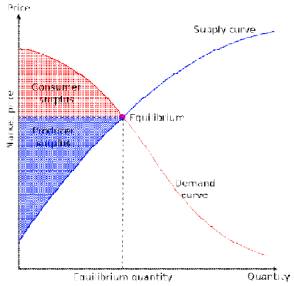


Sports: Plays, strategies, assignments



Physics: Free Bodies

Energy cycle

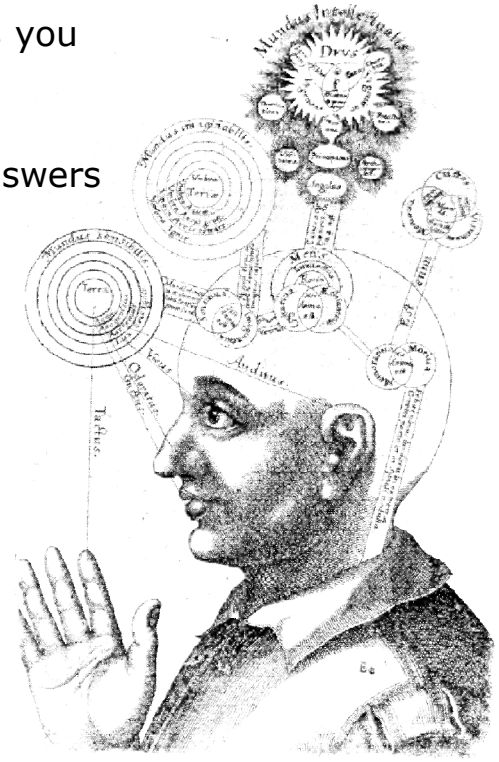


Econ: Supply & demand

Accounting: Debits & credits

Example: is ESP possible?

- How do you organize the world that guides you to your answer?
- Different conceptual models → different answers
- Similar questions
  - Is there fate?
  - Is there luck?
  - Are there curses?



With and without

Without

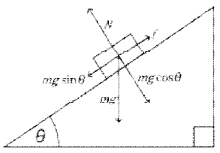
- Just a bunch of raw phenomena
- Doesn't last long -- you will build a model quickly
- But: will it be an effective model?
- Why does it rain? Can witch doctors control it by dancing?
- "The program gets tired and crashes"

With

- Organizes the phenomena you encounter
- Helps you anticipate
- Helps you analyze

Conceptual model of software architecture

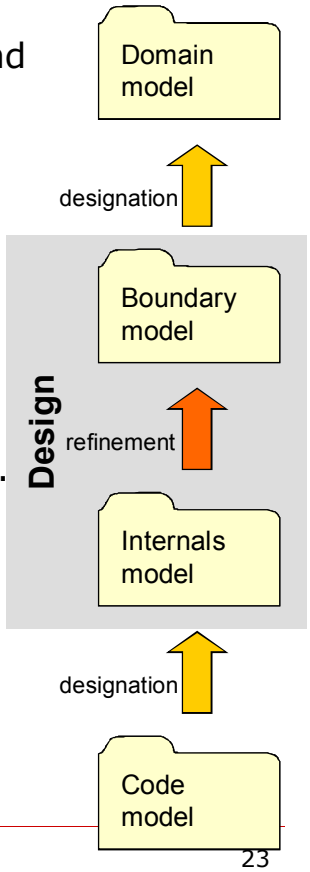
- Model relationships
  - Views & viewtypes
  - Designation
  - Refinement
- Canonical model structure
  - Domain model
  - Design model
    - Internals model
    - Boundary model
  - Code model
- Quality attributes
- Design decisions
- Tradeoffs
- Responsibilities
- Constraints (guide rails)
- Viewtypes
  - Module
  - Runtime
  - Allocation



- Module viewtype
  - Modules
  - Dependencies
  - Nesting
- Runtime viewtype
  - Components
  - Connectors
  - Ports
- Allocation viewtype
  - Environmental element
  - Communication channels

Canonical model structure

- A domain model expresses the intentions, concepts, and workings of the domain.
  - Omits references to the system to be built
  - Is a bridge between engineers and domain experts
- A boundary model expresses the capabilities of the system.
  - Centerpiece is the system to be built
  - Focus on system capabilities, not design
  - There is a single top-level boundary model
- An internals model expresses the design of the system.
  - Refines a boundary model
  - Describes assembly of components that conform to boundary specification
- A code model expresses the solution, either as source code or an equivalent diagram
  - Some design intent lost in code model



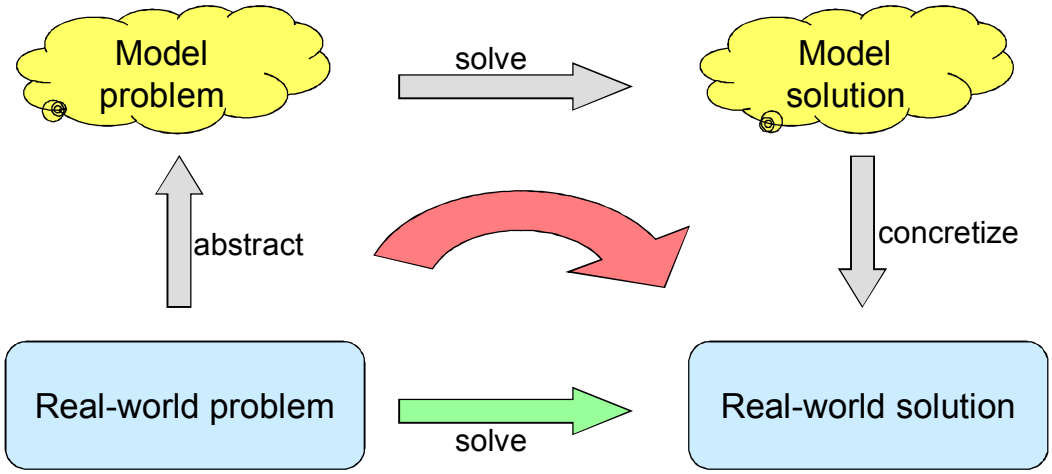
Views and styles

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
- **Architectural styles**
  - Big ball of mud
  - Client-server
  - Pipe-and-filter
  - Map-reduce
  - N-tier
  - …

Engineering with models: Commuting diagram

Mary Shaw's commuting diagram:



"A train is traveling south at 10m/s. Another departs 30 minutes later at 15m/s. When do they meet?"

Talk outline

- Introduction
- Conceptual models
- **Architecture haiku**
- Haiku example: Apache web server
- Group exercise
- Discussion

Architecture presentations: Traditional advice

- You've got 60 minutes to give an architecture presentation. What do you present?

- This will get you started:

- Module view(s)
  - Code organization, layers, DB schemas
- Runtime view(s)
  - Running system + collaborating systems
  - Connections and properties
- Deployment view(s)
  - Hardware, network, topology, etc.

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

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

- Solution description
- Tradeoffs
- Quality attribute priorities
- Architecture drivers (QA scenarios)
- Design rationales
- Constraints
- Architecture styles
- 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 → 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

QA Scenario Templates

- 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")

QA Scenario Examples

- 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

Architecture Drivers

- 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)

Examples

- **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.

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 → portability
  - EJBeans must not start own threads → manageability
  - EJBeans must not write local files → 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

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
- Conceptual models
- Architecture haiku
- **Haiku example: Apache web server**
- Group exercise
- Discussion

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**

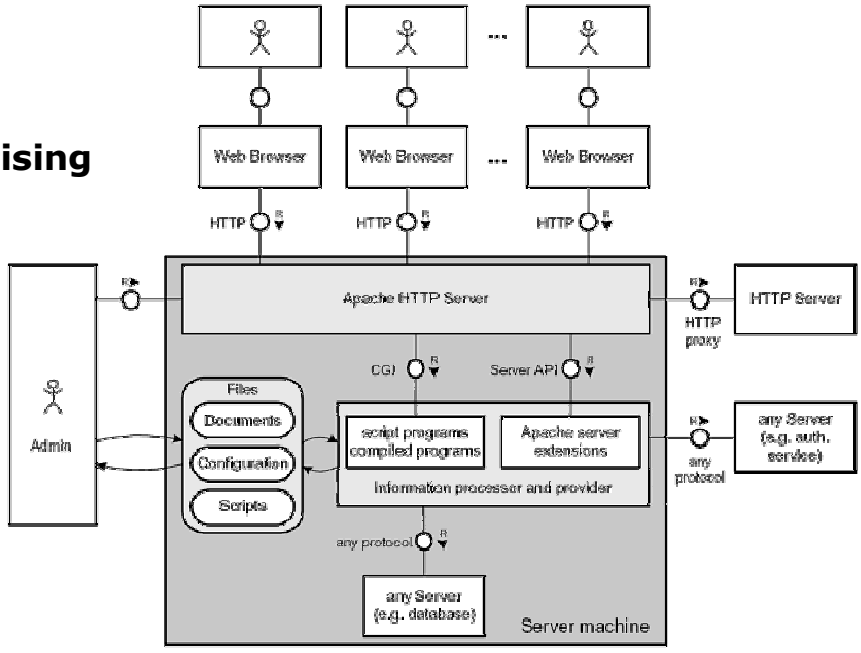


Diagram from: Apache Modeling Project:
Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel, Oliver Schmidt
http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html

Pipe-and-filter diagram

- Request processing
- Apache (dynamically) processes requests
  - Input pipeline
  - Output pipeline
- Note: Runtime viewtype
- **Perhaps worthy to include**

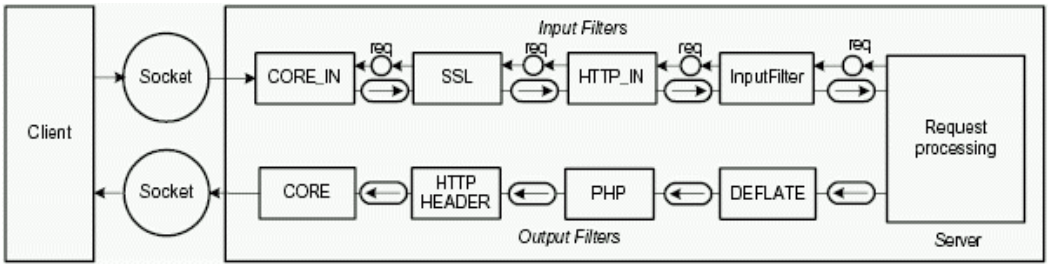


Diagram from: Apache Modeling Project:
Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel, Oliver Schmidt
http://www.fmc-modeling.org/category/projects/apache/amp/Apache_Modeling_Project.html

Talk outline

- Introduction
- Conceptual models
- Architecture haiku
- Haiku example: Apache web server
- **Group exercise**
- Discussion

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
- Conceptual models
- Architecture haiku
- Haiku example: Apache web server
- Group exercise
- **Discussion**

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**

Shameless plugging

- **E-book**
  - One price, 3 formats
  - PDF, ePub, Mobi
  - No DRM
  - RhinoResearch.com $25
- **Hardback**
  - Amazon.com $39.75