Teaching Architecture
Metamodel-First

George Fairbanks

SATURN 2014
7 May 2014

Rhino Research
Software Architecture Consulting and Training
http://RhinoResearch.com
Introduction

About me
- I’ve been teaching OOAD, design, and architecture since the late 90’s
- I wrote a book on architecture
- Still find teaching architecture very difficult

This talk
- This talk describes an approach that’s new to me

My OOAD experience
- Looking for ideal sequence of topics
- Linearizing a web is hard
- Became pretty good at teaching OOAD
  - Many courses, teachers
  - Programming in the small (PITS)
- Teaching architecture is different, harder
  - Few courses, teachers
  - PITL
What’s in this for you?

For teachers
• Teach-the-teacher
• Generalize its pedagogical strategies
• Enough detail here to recreate my course

For newcomers to architecture
• The actual lessons + book references
• Commiseration and a peek behind the curtain
  • Yeah, we’re struggling to teach this topic
Talk summary

Obstacles
- Low motivation
- Abstract ideas
- Big investment before big payoff
- Wrong details
- PITS vs PITL

Pedagogical strategies
- Fail fast
- Make tangible
- Teach small, common task
- Teach metamodel-first
- Heavy on exercises

Learning points
- Q1M2:
  Question first, model second
- Specific/general tradeoff
- Choose one:
  Module, runtime, xor allocation
- No: One diagram to rule them all
- Avoid: "Block diagram"
- Include a full legend
- Eliminate unneeded details
- No: Non-semantic cruft
- No: Naming on technology
- No: Arrowheads (mostly)
Teaching Architecture is Hard
Obstacle: Low motivation

- Developers are not highly motivated to learn architecture
- **Cup already full**: Developers think they know it already
- Many other short-term payoff options (e.g. learn new language)
- **Twitter generation**: Can’t learn calculus by hanging out with math folks
Strategy: Fail fast

Obstacle: Low motivation

• Developers are not highly motivated to learn architecture
• Cup already full: Developers think they know it already
• Many other short-term payoff options (e.g. learn new language)
• Twitter generation: Can’t learn calculus by hanging out with math folks

Strategy: Fail fast

• Headline writing story
• Failure 1: Show that their diagrams stink.
• Failure 2: Focus on problem, not diagram, even in course.
Obstacle: Abstract ideas

- Architecture concepts are largely intangible, abstract
- Some seek abstractions, some seek concrete expression
Strategy: Make the abstract tangible

Obstacle: Abstract ideas

• Architecture concepts are largely intangible, abstract
• Some seek abstractions, some seek concrete expression

Strategy: Make the abstract tangible

• Course scope
  • Teach diagramming, not analysis.
• Course sequence
  • Heavy on exercises
  • Concrete/Abstract/Concrete progression
Obstacle: Big investment before big payoff

- Architecture ideas form a complex web
- Must internalize the ideas before applying them
- Generally takes years to master
Strategy: Teach a small, common task

Obstacle: Big investment before big payoff
- Architecture ideas form a complex web
- Must internalize the ideas before applying them
- Generally takes years to master

Strategy: Teach a small, common task
- Diagrams topic is tiny compared to architecture
- Alternative: Teach from failure examples, e.g. comp.RISKS
Obstacle: Wrong details

- Novices mix abstraction levels
- Novices omit critical details
- Example of drawing dog
Obstacle: Wrong details

- Novices mix abstraction levels
- Novices omit critical details
- Example of drawing dog
Obstacle: Wrong details

- Novices mix abstraction levels
- Novices omit critical details
- Example of drawing dog
Obstacle: Wrong details

- Novices mix abstraction levels
- Novices omit critical details
- Example of drawing dog
Strategy: Teach metamodel-first

Obstacle: Wrong details
- Novices mix abstraction levels
- Novices omit critical details
- Example of drawing dog

Strategy: Teach metamodel-first
- **Just enough theory** to succeed at concrete task
- Tantalizing glimpse of full **cognitive model**
Obstacle: PITS vs. PITL

Obstacle: PITS vs. PITL

- PITS, PITL = Programming in the Small, Large
- PITS: Small examples fit into course neatly
- PITL: Architecture examples, not so neatly
Strategy: Lots of exercise time

Obstacle: PITS vs. PITL

- PITS, PITL = Programming in the Small, Large
- PITS: Small examples fit into course neatly
- PITL: Architecture examples, not so neatly

Strategy: Lots of exercise time

- Spend 60%+ of time on exercises
- General pedagogy: I do, we do, you do
  - Group exercise; I do exercise with class; they repeat
Summary of obstacles and strategies

• Low motivation → **Fail fast**

• Abstract ideas → **Make tangible**

• Big investment before payoff → **Teach small, common task**

• Wrong details → **Teach metamodel (ie legend) first**

• PITS vs PITL → **Lots of exercise time**
Course Outline
Overall course outline

Better Software Diagrams Course

Part 1: Student exercise (fail fast) [30 mins]
- Hello, agenda, goals
- Work in groups of 2-3
- Explain your neighbor’s diagram; I pre-fetch lecture topics

Part 2: Lecture (boring, abstract stuff) [30 mins]
- Goal: 5% think like the coach instead of the rookie
- List of topics, low conceptual model sophistication
- As a group, fix an example diagram using topics

Part 3: Student exercise (apply abstract stuff) [30 mins]
- Decide on Q1M2
- Fix original example
- Reflect on lessons, discuss
Part 1 Exercise: Build a diagram

Design a Library System

The system enables libraries to check books in/out using barcode scanner.

Challenges

- Must tolerate ~1 hour network outages
- Transactions complete within 250ms

Output

1+ diagrams, readable by other engineers, that explain how your design overcomes the challenges

Notes

- Finish in 15 mins
- Focus on clear expression, even if the design is imperfect.
Part 1 Exercise: Build a diagram

Design a Library System

The system enables libraries to check books in/out using barcode scanner.

Challenges

- Must tolerate ~1 hour network outages
- Transactions complete within 250ms

Output

1+ diagrams, readable by other engineers, that explain how your design overcomes the challenges

Notes

- Finish in 15 mins
- Focus on clear expression, even if the design is imperfect.

Rarely draw more than 1 diagram, despite bold text

Invariably draw a general diagram, hard to see how it addresses the challenges

I say this again 5 mins before end; encourage them to redraw diagram for clear expression.
### Sequencing lecture topics

**Q:** What’s the difference between the **coach** and the **rookie**?

**A:** **Conceptual model of architecture**

- But, can’t overload the students

- So, teach conceptual model gradually:
  1. Basic diagrams course
  2. Advanced diagrams course
  3. General architecture modeling course

---

<table>
<thead>
<tr>
<th>Importance</th>
<th>Conceptual Model Sophistication --&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Q1M2: Why does this diagram exist? (1) Eliminate unnecessary details (1) Use a legend (1) SLAP</td>
<td>Q1M2 (more) (1) Notation semantics (1) Diagram prototypes</td>
</tr>
<tr>
<td>(1) Choose: Module, runtime, xor allocation (1) 1DTRTA (2) Choose details to include (2) Legends reveal metamodel (2) Use metamodel consistently</td>
<td></td>
</tr>
<tr>
<td>(2) Modeling --&gt; Engineering (1) Specific/general tradeoff (1) Simple notation</td>
<td>(1) Block Diagram (2) Diagrams are Models</td>
</tr>
<tr>
<td>(2) Modeling --&gt; Engineering (1) Specific/general tradeoff (1) Simple notation</td>
<td>(3) Simplify diagram with refinement (20 Open-Closed Semantics (3) Simplify diagram with views</td>
</tr>
<tr>
<td>(1) Eye Chart, &lt; 15 items (1) Name for intent, not tech (1) No arrowheads (1) Eye Candy</td>
<td></td>
</tr>
</tbody>
</table>
Lecture topics (first course)

Choosing what diagrams to build
- Question first & model second (Q1M2) [JESA: Chap 6.6]
- Specific/general tradeoff [JESA: Chap 15.2.1]

What kinds of diagrams exist?
- Choose one: Module, runtime, xor allocation [JESA: Chap 9.6]
- Prototypical diagram examples [JESA: Chap 4.1]
- Avoid: One diagram to rule them all [JESA: Chap 15.2]
- Avoid: "Block (i.e., generic) diagram"

What's included and excluded?
- Include a full legend [JESA: Chap 15.4]
- Eliminate unneeded details (Q1M2) [JESA: Chap 6.6]
- Avoid: Non-semantic cruft [JESA: Chap 15.4]
- Avoid: Naming based on technology [JESA: Chap 15.1.6]
- Avoid: Arrowheads (mostly) [JESA: Chap 15.4]
Reflection
Reflection

• Overall
  • Effective in short period of time
  • Positive feedback
  • Interest in more

• Mission creep:
  • Really need to add time dimension to the “3 types of diagrams”
  • Add slides to pre-empt problem in last class

• Students quickly backslide
  • Mentoring and follow-up essential
  • But they use legends

• Poor summary: “Use a legend”
  • Compare to golden rule
  • Added this as last slide in course
Talk summary

Obstacles
• Low motivation
• Abstract ideas
• Big investment before big payoff
• Wrong details
• PITS vs PITL

Pedagogical strategies
• Fail fast
• Make tangible
• Teach small, common task
• Teach metamodel-first
• Heavy on exercises

Learning points
• Q1M2: Question first, model second
• Specific/general tradeoff
• Choose one:
  Module, runtime, xor allocation
• No: One diagram to rule them all
• Avoid: "Block diagram"
• Include a full legend
• Eliminate unneeded details
• No: Non-semantic cruft
• No: Naming on technology
• No: Arrowheads (mostly)
About me (George Fairbanks)

- PhD Software Engineering, Carnegie Mellon University

- Program chair: SATURN 2012; Former program committee member: WICSA, ECSA, ICSM, CompArch

- Thesis on frameworks and static analysis (Garlan & Scherlis advisors)

- Architecture and design work at big financial companies, Nortel, Time Warner, others

- Teacher of software architecture, design, OO analysis / design

E-book on Google play store
Hardback on Amazon, etc.