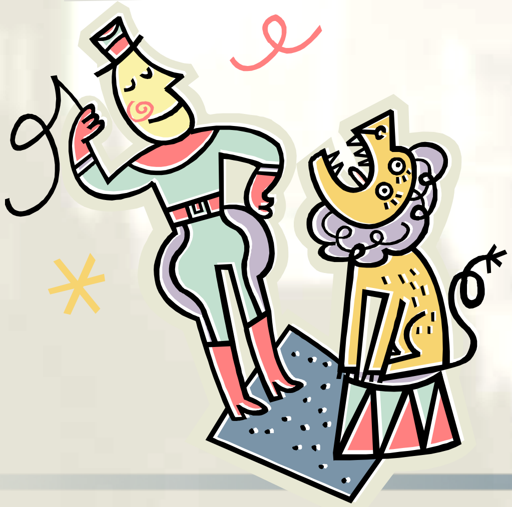


Software Architecture is Bogus

Eric M. Dashofy

February 27, 2013

On Bogosity

from the Jargon File[1]

- **bogus** adj.
  - 1. Non-functional. "Your patches are bogus." 2. Useless. "OPCON is a bogus program." 3. False. "Your arguments are bogus." 4. Incorrect. "That algorithm is bogus." 5. Unbelievable. "You claim to have solved the halting problem for Turing Machines? That's totally bogus." 6. Silly. "Stop writing those bogus sagas."
- **bogosity** *boh-go's*-tee*; n.
  - 1. The degree to which something is bogus.
- **bogon** *boh'gon* n.
  - 1. The elementary particle of bogosity (see *quantum bogodynamics*).

[1] http://www.catb.org/jargon/html/

On Bogosity

from the Jargon File[1]

- **bogus** adj.
  - 1. Non-functional. "Your patches are bogus." 2. Useless. "OPCON is a bogus program." 3. False. "Your arguments are bogus." 4. Incorrect. "That algorithm is bogus." 5. Unbelievable. "You claim to have solved the halting problem for Turing Machines? That's totally bogus." 6. Silly. "Stop writing those bogus sagas."
- **bogosity** *boh-go's*-tee*; n.
  - 1. The degree to which something is bogus.
- **bogon** *boh'gon* n.
  - 1. The elementary p *bogodynamics*).

Special no-prize to anyone who can name the units with which bogosity is measured.

[1] http://www.catb.org/jargon/html/

Questions to Explore

- Why is software architecture bogus?
- Is the bogosity of software architecture *essential* or *accidental*?
- Are there any parts of software architecture that are not bogus?
- Should you drop this class now?

Scope

- Bogus:
  - The idealized, academic perspective of software architecture taught in schools and written about at conferences
    - Extensive model-based designs, documentation, visualization, analysis, architecture-based implementation, model-driven architecture…
- Not (completely) bogus:
  - Good software design done with or without the trappings of formal "software architecture"

Discussion

- How many of you have built software?
- How many of you build models beforehand?
  - Sophisticated models?
  - Models that let you predict or prove properties of the software?
  - Models that can be automatically analyzed with tools?
- How many of you use architectural tools that make the architecture explicit?
- How many projects have an explicit, verifiable architectural style?

Why is software architecture bogus?

- Key thesis: the costs of software architecture outweigh its benefits
    - Modeling and detailed design are extremely costly
    - They, in practice, have a small number of uses
    - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
    - Our most celebrated computer systems credit other things for their success
    - Extraordinarily successful systems seem to have terrible architectures
    - Nobody knows how to evaluate an architecture to learn anything useful

Why is software architecture bogus? (2)

- Modern software is all about middleware
  - It makes most of the key decisions for you
    - Fighting your middleware is stupid, so don't
- All software architecture tools are crap
  - Seriously, I'm not kidding

Why is software architecture bogus?

- Key thesis: the costs of software architecture outweigh its benefits
  - Modeling and detailed design are extremely costly
  - They, in practice, have a small number of uses
  - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
  - Our most celebrated computer systems credit other things for their success
  - Extraordinarily successful systems seem to have terrible architectures
  - Nobody knows how to evaluate an architecture to learn anything useful

Defining Software Architecture

- "The set of principal design decisions about a system."
  - Why this, and why so broad?
    - Comports with the assumption that every system has an architecture.
    - Acknowledges the reality that architecture is more about what is important in a system and less about structure and abstract notions of "elements and form."
    - Not only are architectures different from system-to-system, but META-architectures (e.g., the types of design decisions you even care about) will be different

The Devil of Bogosity™ is in the Details

- There is tremendous variation in the content of the design decisions
  - Across projects within domains, across domains themselves
- So while I can talk rigorously about the concepts…
  - What can I do for architects facing the problem of design in the face of such variation?

Modeling is Costly

- Especially so-called "detailed design"
  - "Hey, let's document everything in a messy graphical notation that we can't programmatically verify or execute!"
    - "And we have to manually translate it to code, and then keep it consistent with the code we write!"
      - And after we do that, we'll probably completely ignore it!
    - "And it will be 60% as difficult to write as our code, only we won't be able to see when we make a mistake!"

Why do we make blueprints for buildings?

- (discuss!)

Why do we make blueprints for buildings?

- To capture design decisions
- To communicate about them among stakeholders
- To determine if the building will be satisfactory to the new owners
- To determine if the building will meet code
- To determine if there are any problems that will derail construction
- To estimate the amount of materials and construction that are needed, which translates to cost

Why do we make blueprints for software?

- To capture design decisions
- To communicate about them among stakeholders
- ~~To determine if the software will be satisfactory to the new owners~~
- ~~To determine if the software will satisfy constraints~~
- ~~To determine if there are any problems that will derail construction~~
- ~~To estimate the amount of effort and construction that are needed, which translates to cost~~

Models Aren’t Useful

- The most powerful models will be the most domain-specific
  - But remember, meta-architecture (the kinds of design decisions) varies from project to project
  - So the applicability of an architectural approach for a project is…
    - One project!
- Do you have time to invent or adapt an architectural approach for each project?
  - Might be useful in a product-line context, but that scope is not much bigger
- How good do you think architectural tools and techniques are going to be if they are only going to be used on one product/product line?

Architecture Astronauts

- “When you go too far up, abstraction-wise, you run out of oxygen. Sometimes smart thinkers just don't know when to stop, and they create these absurd, all-encompassing, high-level pictures of the universe that are all good and fine, but don't actually mean anything at all.”

  -- Joel on Software

http://www.joelonsoftware.com/articles/fog0000000018.html

Hammer Factory Factories

- "Yup. So we stopped selling those schematics and started selling hammer-factory-building factories. Each hammer factory factory is built for you by the top experts in the hammer factory factory business, so you don't need to worry about all the details that go into building a factory. Yet you still get all the benefits of having your own customized hammer factory, churning out your own customized hammers, according to your own specific hammer designs."

  -- Joel on Software

http://discuss.joelonsoftware.com/default.asp?joel.3.219431.12

(Consider the source)

- Designer of Excel VBA
- Runs a company whose software is written in a proprietary language (Wasabi) that is run through a proprietary generator to generate ASP and PHP to target multiple platforms

But he has a point...

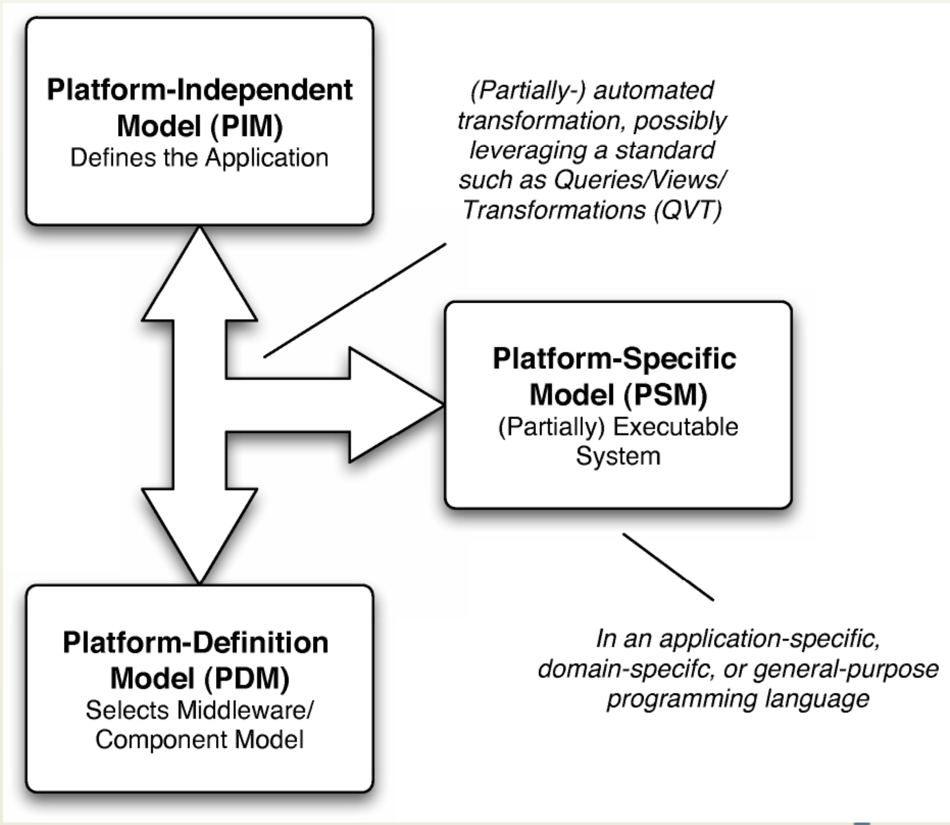
- "All problems in computer science can be solved with another layer of abstraction…except too many layers of abstraction."
  - How much abstraction is too much?
  - The architecture discipline rarely says "stop"
  - Intense investment and marketing in frameworks that don't pan out
    - Jini, JXTA, Microsoft Hailstorm…
  - "Not invented here" means many frameworks that have only superficial differences

How Models Age

- "You see, this profession is filled to the brim with unrealistic [models]. [Models] who thought their [redacted] would age like wine. If you mean it turns to vinegar, it does. If you mean it gets better with age, it don't."
- Code has to change. Models don't.
  - The bigger the model, the more expensive it is to change.

But but but MDA!

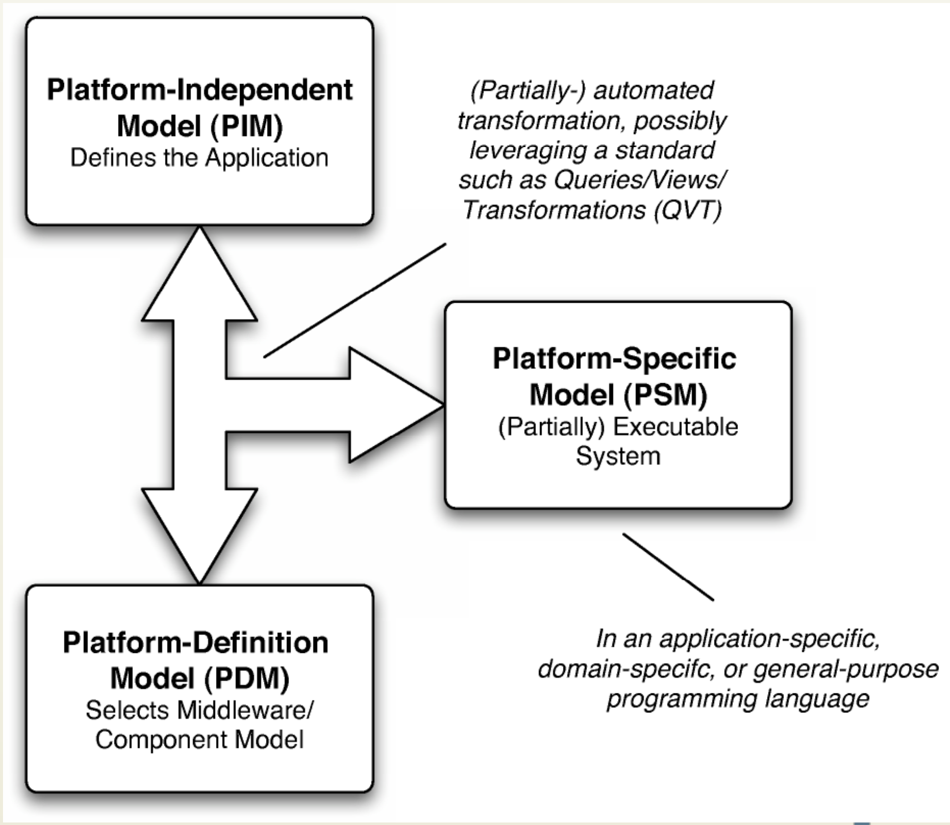
- What is the difference between model-driven architecture and the Lord of the Rings?


Platform-Independent Model (PIM)
Defines the Application
(Partially-) automated transformation, possibly leveraging a standard such as Queries/Views/Transformations (QVT)
Platform-Specific Model (PSM)
(Partially) Executable System
Platform-Definition Model (PDM)
Selects Middleware/ Component Model
In an application-specific, domain-specifc, or general-purpose programming language

Redrawn from the MDA documentation

But but but MDA!

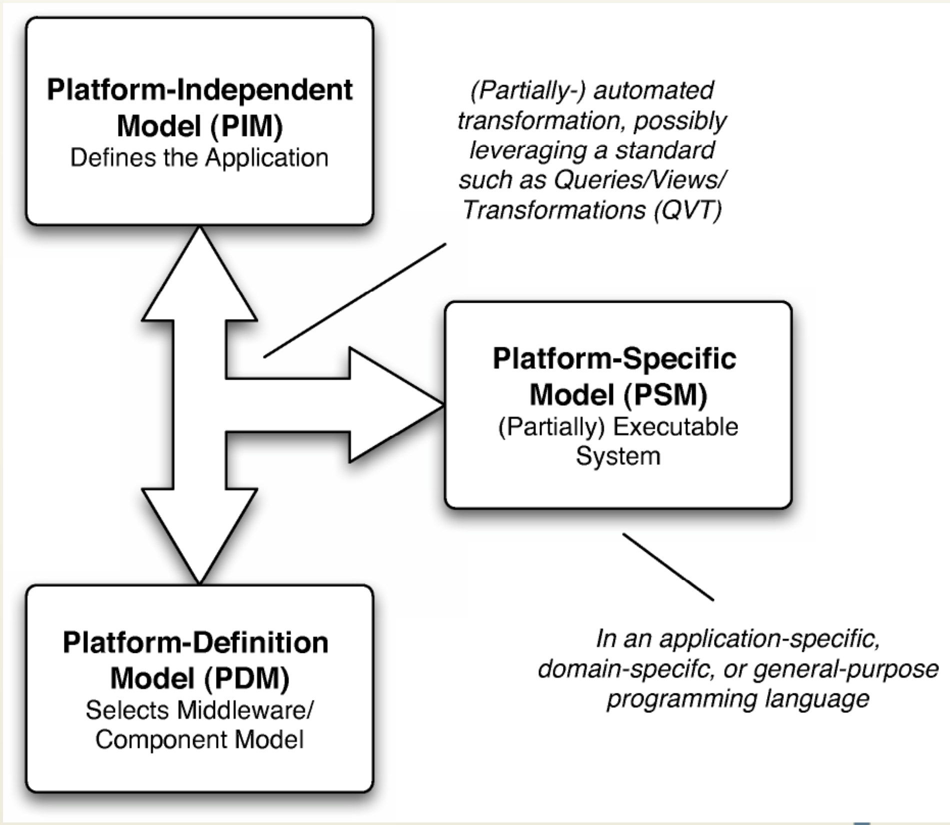
- What is the difference between model-driven architecture and the Lord of the Rings?
  - One is a fantasy set in a world where wizards and other unlikely characters save the world against impossible odds.


Platform-Independent Model (PIM)
Defines the Application
(Partially-) automated transformation, possibly leveraging a standard such as Queries/Views/ Transformations (QVT)
Platform-Specific Model (PSM)
(Partially) Executable System
Platform-Definition Model (PDM)
Selects Middleware/ Component Model
In an application-specific, domain-specifc, or general-purpose programming language

Redrawn from the MDA documentation

But but but MDA!

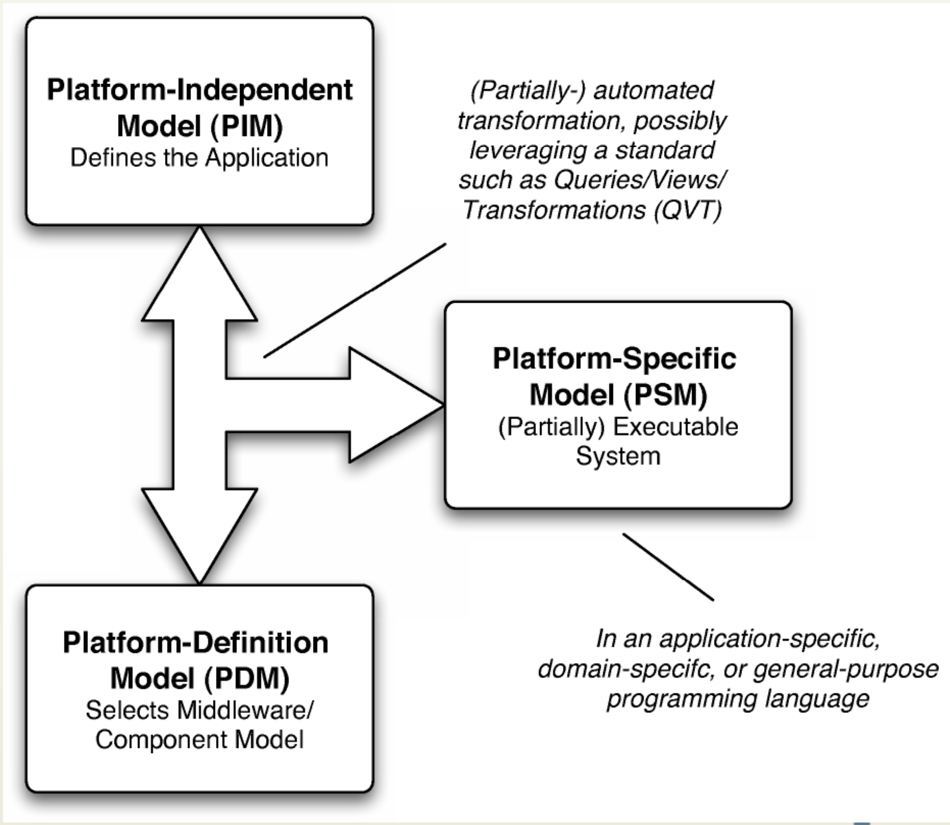
- What is the difference between model-driven architecture and the Lord of the Rings?
  - One is a fantasy set in a world where wizards and other unlikely characters save the world against impossible odds
  - The other, of course, involves orcs.


Platform-Independent Model (PIM)
Defines the Application
(Partially-) automated transformation, possibly leveraging a standard such as Queries/Views/ Transformations (QVT)
Platform-Specific Model (PSM)
(Partially) Executable System
Platform-Definition Model (PDM)
Selects Middleware/ Component Model
In an application-specific, domain-specifc, or general-purpose programming language

Redrawn from the MDA documentation

But but but MDA!

- Round-tripping is impossible to implement well
- For MDA to be useful, the model has to have at least one big payoff OTHER than the ability to generate code
- Probably works well in some domain-specific applications, but requires massive investment and suffers same scope problems as other approaches


Platform-Independent Model (PIM)
Defines the Application
(Partially-) automated transformation, possibly leveraging a standard such as Queries/Views/ Transformations (QVT)
Platform-Specific Model (PSM)
(Partially) Executable System
Platform-Definition Model (PDM)
Selects Middleware/ Component Model
In an application-specific, domain-specifc, or general-purpose programming language

Redrawn from the MDA documentation

Why is software architecture bogus?

- Key thesis: the costs of software architecture outweigh its benefits
  - Modeling and detailed design are extremely costly
  - They, in practice, have a small number of uses
  - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
  - Our most celebrated computer systems credit other things for their success
  - Extraordinarily successful systems seem to have terrible architectures
  - Nobody knows how to evaluate an architecture to learn anything useful

Beware Architectural Langoliers!

- Technology is moving faster than ever
  - This is not likely to change, and is actually likely to get worse: *hyper-exponential change*
- Building tools and methods and notations that make construction more efficient takes **time and effort**
  - From the definition, we know they are of limited horizontal use
- By the time we build even the most basic tools, will they even be relevant?
  - Are we building tools or software?

The Half-Life of Architecture

- If you choose an architectural approach or framework…
  - How many projects does it have to be good for in order to pay off?
  - How many projects is it likely to be good for in reality?

Why is software architecture bogus?

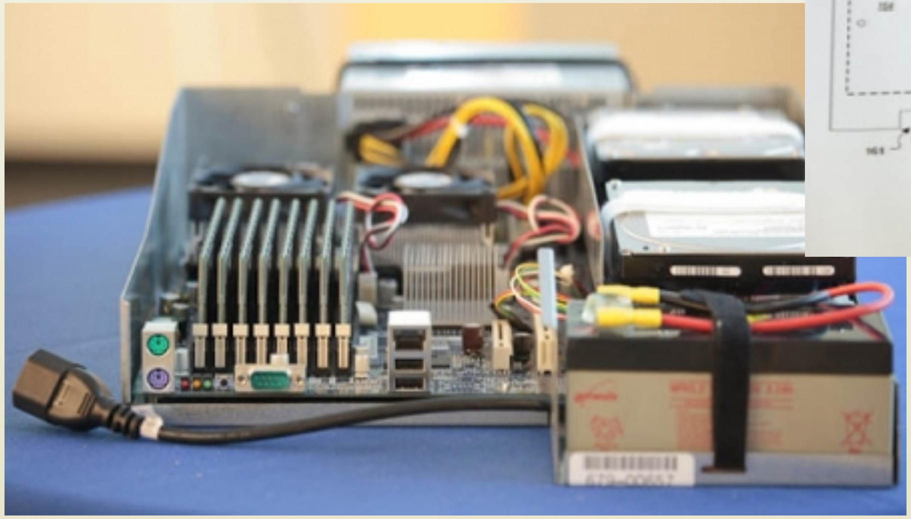
- Key thesis: the costs of software architecture outweigh its benefits
    - Modeling and detailed design are extremely costly
    - They, in practice, have a small number of uses
    - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
    - Our most celebrated computer systems credit other things for their success
    - Extraordinarily successful systems seem to have terrible architectures
    - Nobody knows how to evaluate an architecture to learn anything useful

It is a Mystery

- What is this? Where is it?

Inside a Google Data Center

- Is this architecture? Is it *software* architecture?
- What is the software architecture of the sophisticated monitoring system?
  - Is there one?



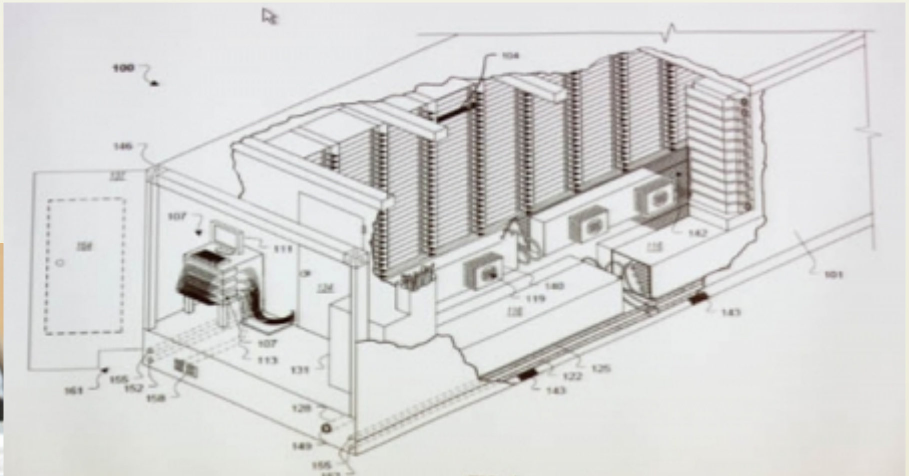


FIG. 1

What is Google's Software Architecture?

- Step 1: MapReduce, BigTable
- Step 2: Glue code
- Step 3: MASSIVE PROFIT

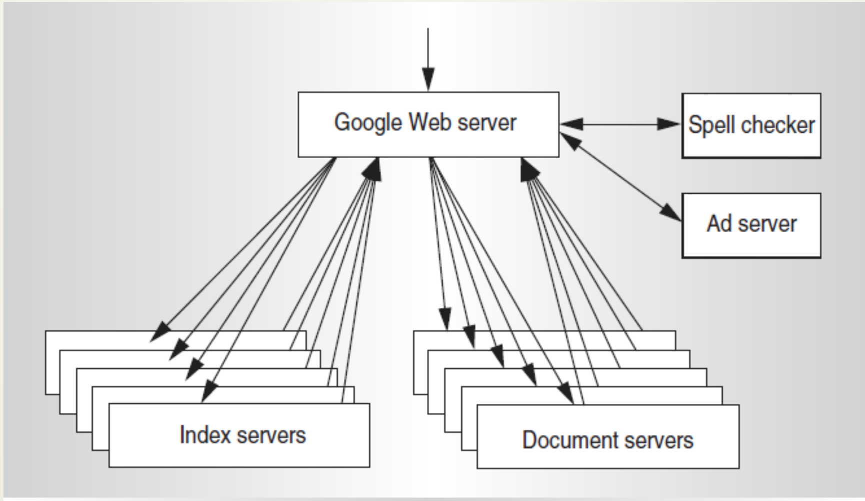


Figure 1. Google query-serving architecture.

- "Google's software architecture arises from two basic insights. First, we provide reliability in software rather than in server-class hardware, so we can use commodity PCs to build a high-end computing cluster at a low-end price. Second, we tailor the design for best aggregate request throughput, not peak server response time, since we can manage response times by parallelizing individual requests."
  - --Barroso et al. *Web Search for a Planet: The Google Cluster Architecture*

What is Facebook’s Software Architecture?

- …

What is Facebook's Software Architecture?

- HAHAHAHAHAHAHAHAHAHAHA!

AND THEN I SAID...
HEY, WAIT A MINUTE, SHOULDN'T WE SEE IF THIS WORKS BEFORE WE DEPLOY IT?

What about the Web?

- REST makes the Web successful!
  - Right? right? hello?
- Guiding principle: separation of resources from representation
  - How many Web developers keep this in mind while developing?
  - How often do we really see the same URL serving different representations of the same resource?

What about the Web? (2)

- Five key REST constraints
  - 1. Client-server
    - Servers do not store user state, right?
  - 2. Stateless communication
    - Try browsing without cookies or iframes or AJAX
  - 3. Cacheable
    - Straightforward optimization of HTTP/1.0
  - 4. Layered system
    - How many intermediate HTTP proxies do you use that are not your company’s mandated firewall?
  - (cont.)

What about the Web? (3)

- Five key REST constraints (cont).
  - 5. Uniform Interface
    - Identification of resources
      - What resource does http://amzn.to/f5Hnc7 identify?
    - Manipulation of resources through these representations
      - Or, you know, completely separate web forms that hit a MySQL database…
    - Self-descriptive messages
      - Syntactic, not semantic. Semantic descriptions required for…
    - Hypermedia as the engine of application state
      - All the best Web applications are built as giant finite-state automata with hyperlinks representing explicit transitions from one state to the other, right?
      - How's that back button working for ya?

Why is software architecture bogus?

- Key thesis: the costs of software architecture outweigh its benefits
  - Modeling and detailed design are extremely costly
  - They, in practice, have a small number of uses
  - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
  - Our most celebrated computer systems credit other things for their success
  - Extraordinarily successful systems seem to have terrible architectures
  - Nobody knows how to evaluate an architecture to learn anything useful

Let’s do a quick design exercise!

- Let’s design the architecture of a simple modern database-driven Web application!

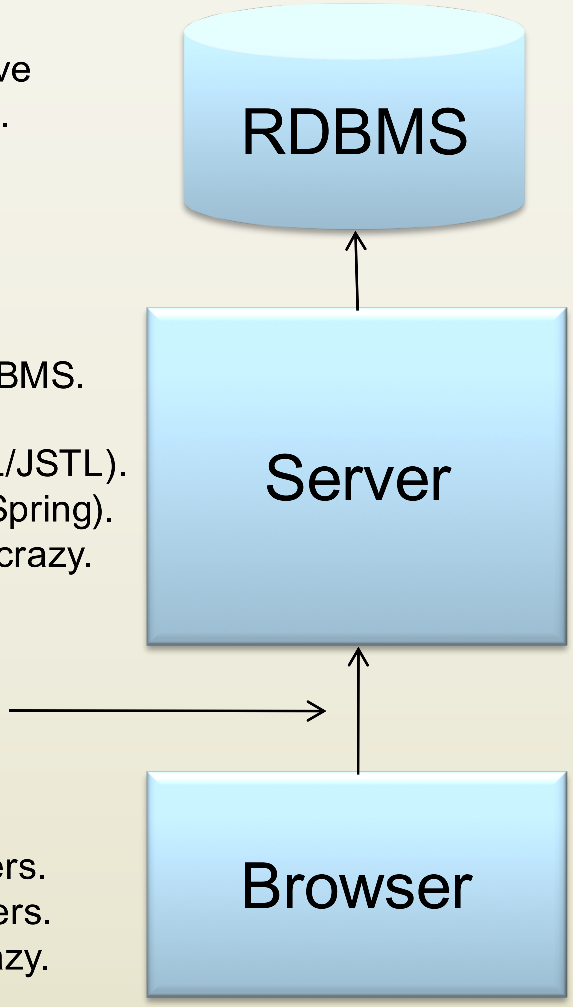
I'm guessing it looks like this…

Some non-standard SQL variant that may or may not have triggers and stored procedures as part of the app code. No REST here at all.

Crazy ORM layer that hides the fact that you have an RDBMS.
Base language (maybe Java).
Completely separate Web dialect of base language (JSP/EL/JSTL).
"Framework" with novel configuration file languages (JSF/Spring).
REST-breaking session manager to keep you from going crazy.

URL-encoded or XML or JSON or custom types or any MIME type you can think of, really.

HTML that doesn't render right in all your browsers.
CSS that definitely doesn't render right in all your browsers.
JavaScript that is only partially supported in some browsers.
JQuery or DOJO or Prototype to keep you from going crazy.
REST-breaking cookies, iframes, AJAX.

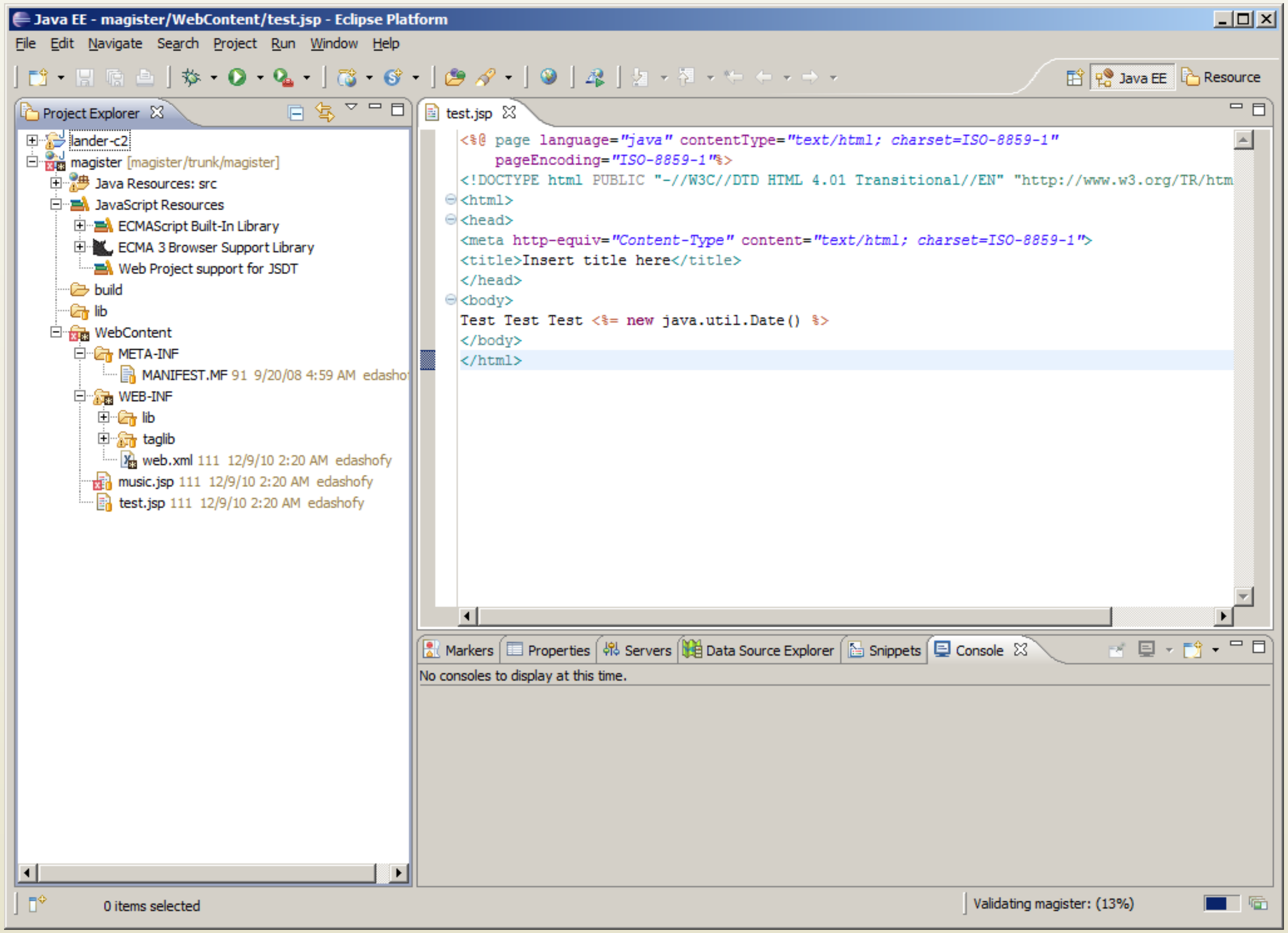


Heroic Architects of the Web

- Dude making a phone book app at CERN
- Brendan Eich and whoever at Netscape decided that the best programming language to make every browser use was one that nobody had ever heard of.[1]
  - "I know only one programming language worse than C and that is Javascript." –Robert Cailliau, the "fifth Beatle" of the WWW
- Lou Montulli, who invented Cookies
  - But did not even write a standard for 3 years…
  - And invented browser tracking, cross-site scripting hacks, cross-site request forgery hacks…

[1] He did, however, make OOPSLA/SPLASH relevant for at least a dozen years as people tried to figure out how to optimize his monstrosity.

Eclipse IDE


Java EE - magister/WebContent/test.jsp - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Java EE
Resource
Project Explorer
test.jsp
Markers
Properties
Servers
Data Source Explorer
Snippets
Console

```
lander-c2
magister [magister/trunk/magister]
  Java Resources: src
  JavaScript Resources
    ECMAScript Built-In Library
    ECMA 3 Browser Support Library
    Web Project support for JSDT
  build
  lib
  WebContent
    META-INF
      MANIFEST.MF 91  9/20/08 4:59 AM  edasho
    WEB-INF
      lib
      taglib
      web.xml 111  12/9/10 2:20 AM  edashofy
    music.jsp 111  12/9/10 2:20 AM  edashofy
    test.jsp 111  12/9/10 2:20 AM  edashofy
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/htm
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
Test Test Test <%= new java.util.Date() %>
</body>
</html>
```

No consoles to display at this time.

0 items selected

Validating magister: (13%)

Eclipse IDE (cont.)

- Uses the OSGI plug-in architecture
  - Each plug-in specifies explicit dependencies
- Each plug-in is a set of Java classes
  - The interface to a plug-in is usually all those classes
  - Pointers and objects are allowed to be passed across plug-ins creating an architectural style called "object soup"
  - Behavioral extensions implemented through an XML-based extension mechanism
  - Lazy loading constraint means plugin's initial UI must be defined in XML
    - But since this is inadequate, let's invent a new language of conditionals, variables, boolean logic in XML

What is driving modern software?

- Is it architecture or marketecture?
  - Increasingly, the market, not sound technical reasoning, influences the design of our systems
    - There are consequences for this!
      - Spam
      - XSS attacks
      - XSRF attacks
      - Platform fragmentation
      - Purposeful integration difficulties
      - Ridiculously complex middleware systems

Why is software architecture bogus?

- Key thesis: the costs of software architecture outweigh its benefits
  - Modeling and detailed design are extremely costly
  - They, in practice, have a small number of uses
  - Large maintenance tail leads to neglect
- Software is changing too rapidly to build artifacts
- Lack of practical benefits
  - Our most celebrated computer systems credit other things for their success
  - Extraordinarily successful systems seem to have terrible architectures
  - Nobody knows how to evaluate an architecture to learn anything useful

Analysis Paralysis

- Recall the blueprints discussion
- Sometimes limited models can give you limited insight about the system
  - Performance models, simulations
- High-value analysis rests on a large number of assumptions
  - That the models are substantially cheaper to develop than the code
  - That you can learn something really valuable or non-intuitive from the models
  - That you can somehow keep those models consistent with the code when you write it
  - That you can somehow keep those models consistent with the code after you write it

Why is software architecture bogus? (2)

- Modern software is all about middleware
  - It makes most of the key decisions for you
    - Fighting your middleware is stupid, so don't
- All software architecture tools are crap
  - Seriously, I'm not kidding

Discussion

- Enumerate some middleware/frameworks you know!

The Rise of Middleware

- Middleware and frameworks in general, rather than "software architecture," increasingly dominate high-level software design
  - RPC: DCE, Courier, CORBA, COM/DCOM/COM+, RMI, XML-RPC, SOAP
  - Messaging: MQSeries, MSMQ, JMS, ESBs
  - Orchestration/Choreography: various ESBs
  - Web backend: Perl, Python, JSP 1.0, JSP+EL+JSTL, Genshi, Django, ClearSilver, Velocity, Struts, Seam, Pylons, ColdFusion, ASP, ASP.NET, PHP, various CMSes, Spring, JSF, Ruby with and without Rails, Groovy + Grails…
  - Web frontend: JQuery, DOJO, GWT, YUI, Prototype
- And you thought the programming language holy wars were bad!

Middleware and Architecture

- Middleware and frameworks induce architectural styles (Di Nitto and Rosenblum, 1999)
  - But not necessarily the style you want
    - Middleware locks you in
  - Other accidental constraints limit the frameworks you can choose
    - Cost, developer market, language support, availability of libraries…
  - Putting your own style on top of these systems is subtle and difficult
- The development of new middleware nearly paces our development of new software systems
  - How many times will one organization reuse a specific fw?

Middleware and Marketecture

- Middleware is proffered by two groups
  - Big companies who want to lock you into their platform
  - Open-source zealots who think they have found the One True Way
- Who is *really* designing your software?
  - Hint: Not you!
- Sad Realities
  - The frameworks or middleware you pick will impose about 75% of the important design decisions on your software without your explicit consent
  - You choose frameworks/middleware early in the project, at the time you're least prepared to do so
  - If you screw up, the costs to change are incredible
  - Your framework will probably not be viable for more than one project even if you pick "right."

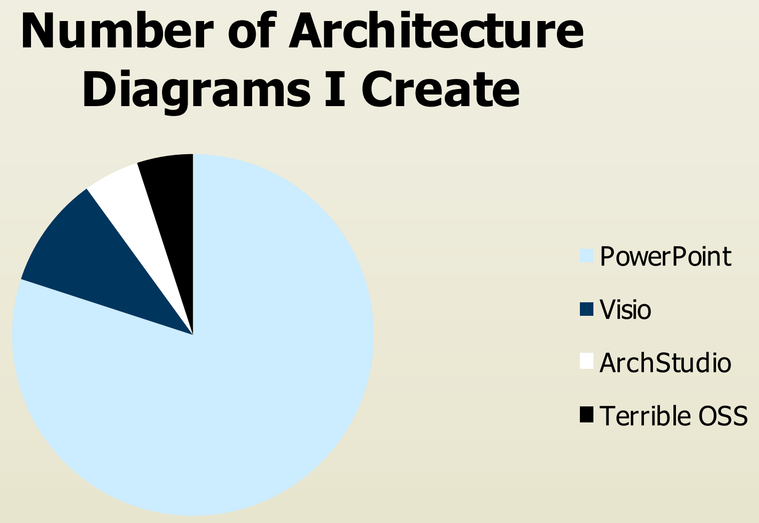
Why is software architecture bogus? (2)

- Modern software is all about middleware
  - It makes most of the key decisions for you
    - Fighting your middleware is stupid, so don't
- All software architecture tools are crap
  - Seriously, I'm not kidding

What tools do people actually use to capture architectures?

- Word, Excel, PowerPoint, Visio (Lots)
- Rational Rose, Rhapsody, other UML (Some)
- Others (System Architect, Enterprise Architect…) (A few)

- What value do these add?



Maturity vs. Evolution

- Hyper-exponential change erodes the foundation for tools
  - Tools are expensive to build and maintain
  - Will you ever have high-quality tools for an approach that is "hot" for six months or a year?
    - Note the phenomenon that you do see high-value tools for programming languages, but almost never for frameworks or modeling approaches

What tools might be promising?

- Model checkers
  - Spin, Alloy, SMV
    - But these are just abstract problem solvers now
  - High-fidelity simulators
    - Which we still have to build as one-offs
  - Performance and comms analysis (Rapide-esque)
    - But the models are more complicated than the code…

What isn't bogus?

- "First, great designs have **conceptual integrity**—unity, economy, clarity. They not only work, they delight, as Vitruvius first articulated." –Fred Brooks, *The Design of Design*
  - Related: the principle of least surprise

Conceptual Integrity

- Key concept: **if you learn something about one part of the system, you should have learned something about the rest of the system**
- The least-bogus concepts from software architecture are those that directly support conceptual integrity
  - Styles
  - Patterns
  - Parsimony
  - An appropriate level of abstraction
- These also require very little tooling, few start-up costs, and can be adapted most easily through refactoring

Also Less Bogus

- Simple, high-level models that fit on a slide or two
  - If you can’t make these or they look like a fully-connected graph, you’re doing it wrong
- Detailed models (e.g., discrete event simulations) that are relatively cheap to produce and tell you interesting things
- Modeling notations (like UML) as a shared symbolic vocabulary for enabling more rigorous conversations
- Aggressive refactoring in support of conceptual integrity
- …your thoughts?

Concluding Thoughts

- Software architecture needs a stronger focus on conceptual integrity: establishing, gauging, maintaining
- The other stuff has potential value but the cost:benefit ratio is out of whack
- Tools have high potential value but are expensive to implement and of limited use with low shelf-life
  - Doomed by hyper-exponential change?
  - Really adaptable tools would be awesome, but are rare
- Some bogosity essential, some accidental

Should you drop this class?

- Only if you have already bought the book (new, not used)
- Arguments for:
  - We still don't know how to teach software design
    - But I have a feeling it's all about conceptual integrity
- Arguments against:
  - You will learn about fundamentals of conceptual integrity
  - You may help drive us toward architecture instead of marketecture in the future