(How to) Use More Open Source

...in your next Software Acquisition

Who we are...

Rob is a director-level software engineer and architect in commercial software, and part of Agile Assisted Acquisitions.

Eric is an experienced developer who mixes technology and policy for 18F, and previously for the Sunlight Foundation.

We are NOT security professionals.

18F

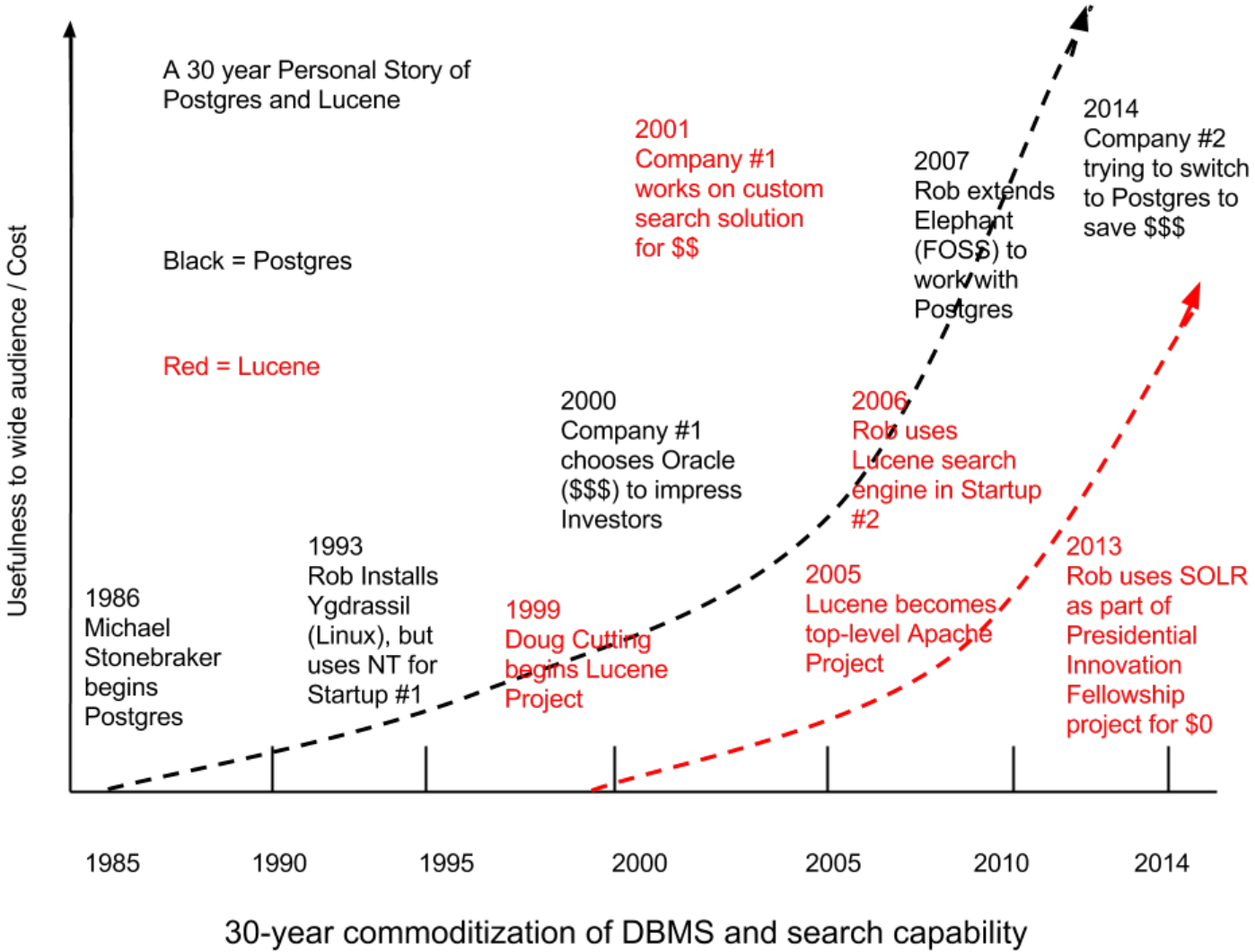
Outline

- Commoditization: An Irresistible Force
- Architecture and Modularity
- Risk Management
- Security
- A Checklist of Commodities
- Understanding the Cloud

18F

A Brief History of Postgres

- It has always been excellent, it has always been usable, but…
- It has inexorably become easier to use and more performant…
- To the point of being well past a tipping point.


A 30 year Personal Story of Postgres and Lucene
Black = Postgres
Red = Lucene
Usefulness to wide audience / Cost
1986 Michael Stonebraker begins Postgres
1993 Rob Installs Ygdrassil (Linux), but uses NT for Startup #1
1999 Doug Cutting begins Lucene Project
2000 Company #1 chooses Oracle ($$$) to impress Investors
2001 Company #1 works on custom search solution for $$
2005 Lucene becomes top-level Apache Project
2006 Rob uses Lucene search engine in Startup #2
2007 Rob extends Elephant (FOSS) to work with Postgres
2013 Rob uses SOLR as part of Presidential Innovation Fellowship project for $0
2014 Company #2 trying to switch to Postgres to save $$$
1985
1990
1995
2000
2005
2010
2014
30-year commoditization of DBMS and search capability

18F

Exploit Commoditization

Don’t pay for that which is now a commodity, and more and more is every year.

Ease-of-use often lags commoditization of function, so don’t be afraid to pay for technical support.

18F

The Coming Open Source Singularity

- The Unix Way made real: write small, independently recombinable programs.
- Made real by GitHub and other code-sharing sites.
- The very nature of programming has changed, and the trend will continue.
- Programmers are now fundamentally more empowered than they used to be---and every year this is 20% more true.
- I spend most of my time figuring out how to reuse code.

18F

Must clearly distinguish...

- That which you may reuse, from
- That which you must develop.

The principles are the same, but in practice they are quite different.

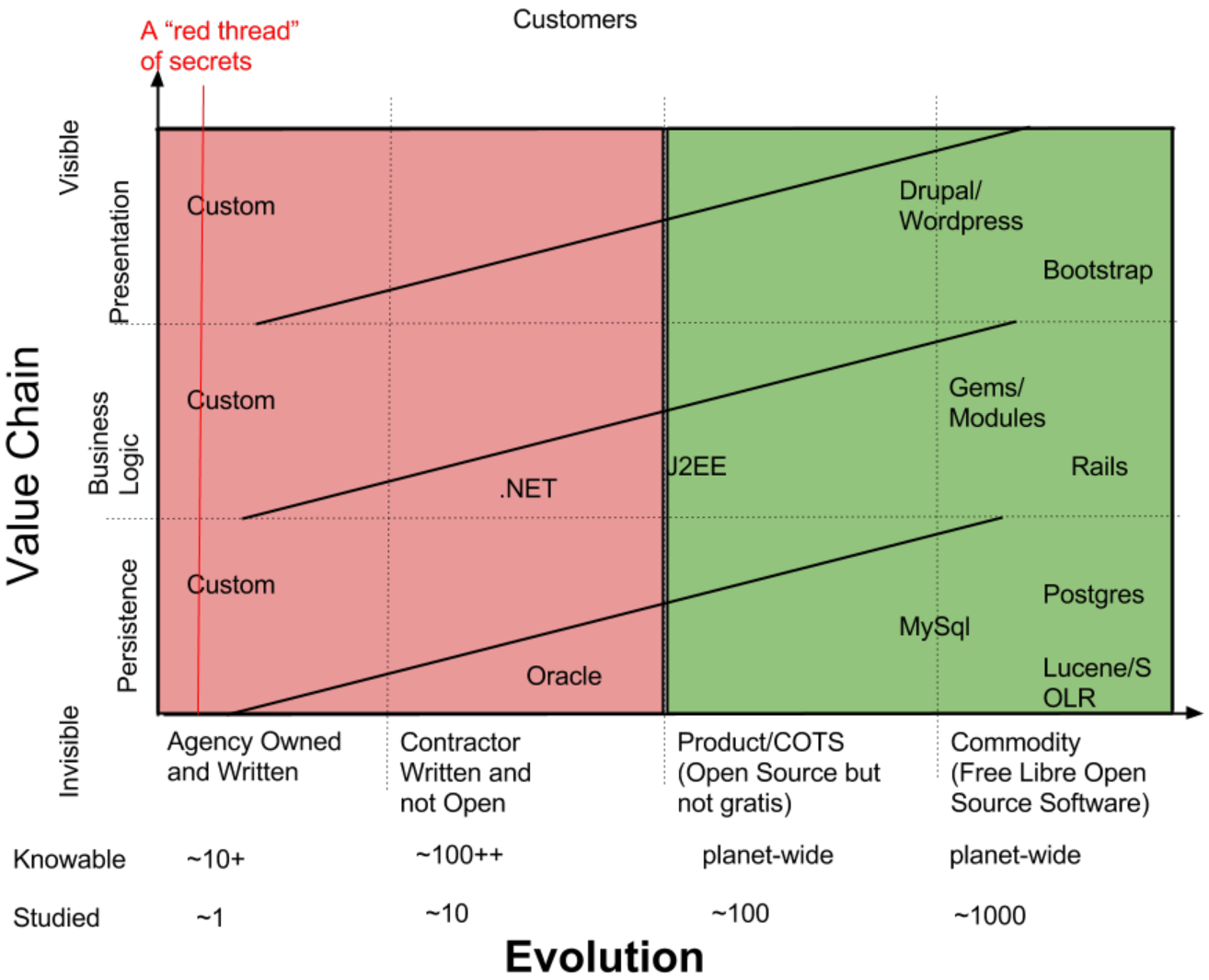
“Buy vs build” but now also “reuse or custom-build”.

18F

Architecture trumps Coding

A well-designed system with good interfaces and bad code beats a hairy system with poor interfaces and brilliant code.

18F


Customers
A “red thread” of secrets
Value Chain
Visible
Presentation
Business Logic
Persistence
Invisible
Custom
Custom
Custom
.NET
Oracle
J2EE
Drupal/ Wordpress
Bootstrap
Gems/ Modules
Rails
Postgres
MySql
Lucene/S OLR
Agency Owned and Written
Contractor Written and not Open
Product/COTS (Open Source but not gratis)
Commodity (Free Libre Open Source Software)
Knowable
~10+
~100++
planet-wide
planet-wide
Studied
~1
~10
~100
~1000
Evolution

18F

“90% of everything is crud.”

-- Sturgeon’s Law.

There is an Ocean of open-source software out there, and 90% of it is crud, and this is irrelevant.

But we must understand how to manage this Ocean of free software.

18F

Use lots of free software

But not indiscriminately. Evaluate the inclusion of software by:

The activity of the community supporting it,

How many lines of code it saves you (don’t include a large project to save a few lines.)

If it has less than 50 contributors, code review it yourself.

Small projects which are easily code reviewed and need not be updated do not represent much of a risk.

If you need to update something frequently then it needs to be be big.

18F

Principles

- Try to stay as evolved as possible.
- Each year, your project requires writing less and less custom code.
- Don’t pay too much for things which are already commoditized.
- Modularity allows you to control the open source evolution.
- Pay for services which are not yet commoditized.
- Try to have only a “red thread” of secrets---thin and clearly delineated.
- Insist on modular replaceability as a risk mitigator.

18F

Security

- Principles
- Reuse of existing projects
- Writing your own
- FISMA tips

18F

Security Principles

- Risks you can see are better than risks you cannot see.
- More eyeballs means decreased risks
- Easier to keep small, changeable secrets---therefore a codebase is a terrible secret

18F

Security: Reuse of Existing Projects

- Either small, code reviewable, and unchanging, or big and highly used…
- Take the money you save from reuse and put it into penetration and code review testing
- Be educated, but not a “shiny object person”
- Understand Release Sweet Spot: Not too new, not to old---but ALWAYS moving

18F

Security: For your own Code

- Open-source from the start
  - Makes for better code
  - Decreases vendor lock-in
  - Let citizens reuse (and, in theory, contribute)
  - Makes transition of project as inexpensive as possible
- Modularity trumps coding

18F

Yourself for Each Module

- Is this something that is potentially reusable?
  - If yes, has someone else already written it?
  - If yes, then it is prime candidate for open source from the first.
- Am I writing something that someone could make a business out of?
  - If yes, why am I doing that?

18F

Q: What makes Open Source Secure?

A: Enough public eyeballs are on it.

Don’t take a project developed as closed-source and make it open-source. Instead, work in the light from the beginning.

18F

Security FISMA

- Try to avoid mixing FISMA-levels in the same system.
- Understand that FISMA-public exists (conceptually.)

18F

Checklist (the Biggest)

1. Content Management Systems (Drupal, WordPress, Github Pages)
2. Full-text Search Engines (SOLR, Elasticsearch, Lucene)
3. GUI frameworks (Bootstrap)
4. Application frameworks (Rails, Django)
   No-SQL Databases (MongoDB, Redis)
5. Relational Databases (Postgres, MySQL)

Commoditization of Unstructured Document Websites

Application

Rocket Science

Structured Data,
Algorithmic Regulations

Unstructured Data, User
Entry of Documents

Barely
Commoditized

Highly
Commoditized

Custom Website

Application Framework
(Django, Rails)

Content Management System
(Drupal WordPress)

Implementation

How to use the Checklist...

Take a good solid afternoon to ask yourself:

What fraction of my project could be carved out and accomplished with these commoditized systems?

(Or stay for our workshop.)

18F

The Cloud

...is just the commoditization of RUNNING computer systems.

18F

Further Reading

- http://ben.balter.com/2014/08/03/why-isnt-all-government-software-open-source/ -- Excellent work by Ben Balter.
- http://pingv.com/node/58 -- diagram of release sweet spot
- http://www.postgresql.org/about/history/ - history of Postgres