(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 founder of Agile Assisted Acquisitions, a part of 18F.

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

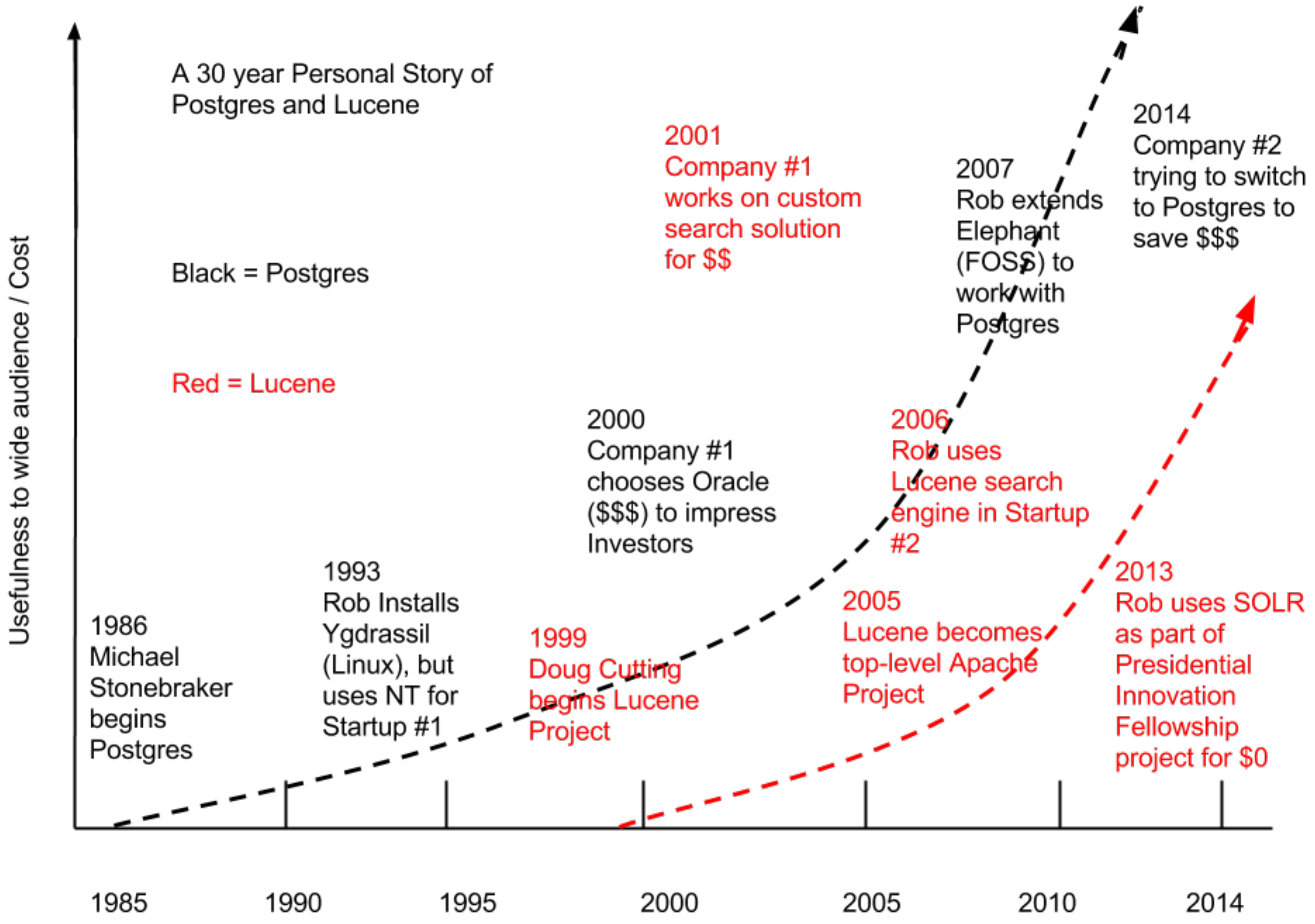
We are NOT security professionals.

Outline

- <u>Commoditization: An Irresistible Force</u>
- Risk Management
- Security

Commoditization: A Brief History of Postgres

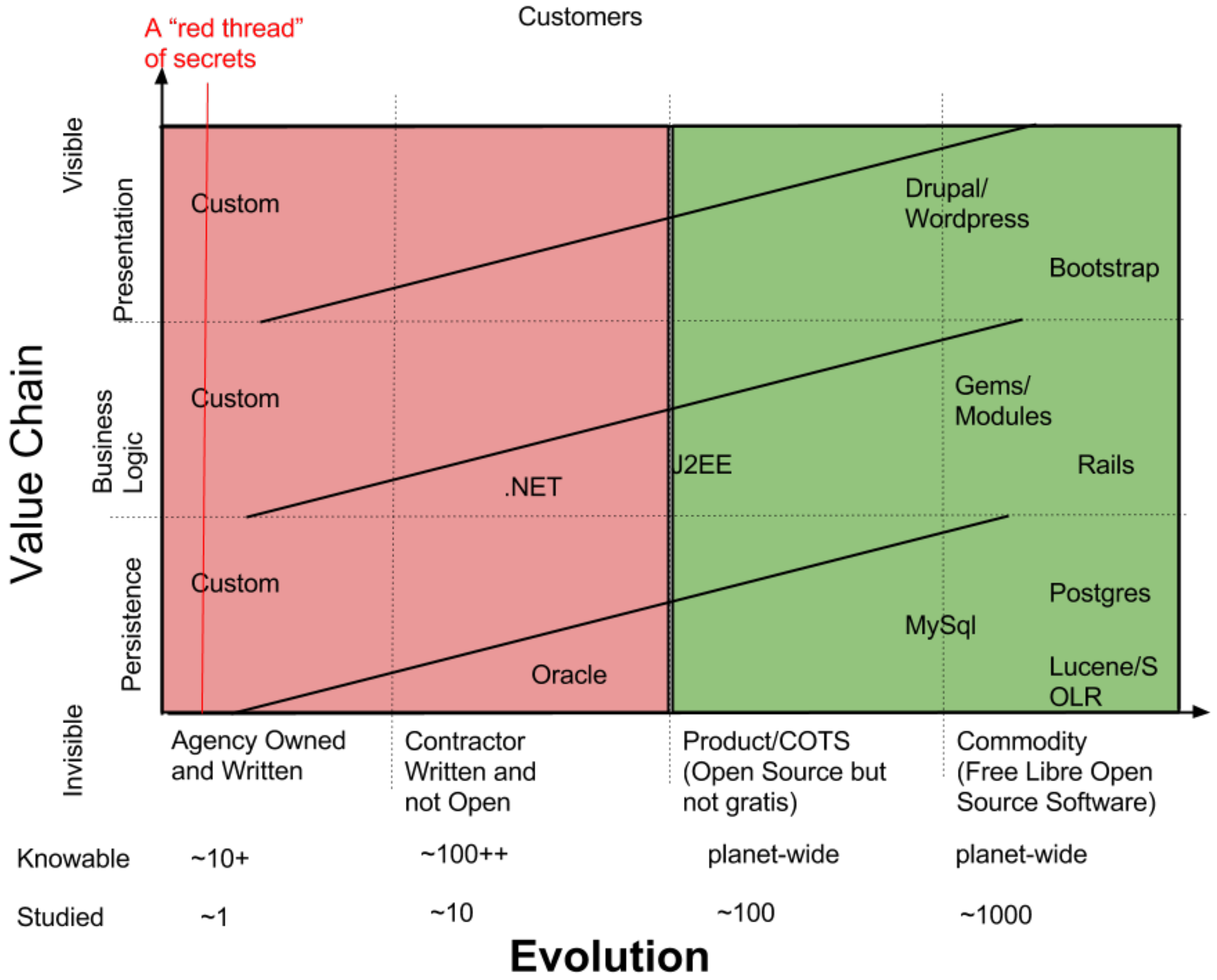
- It has always been excellent, it has always been usable, but…
- It has inexorably become easier to use and more performant…
- And is now past a tipping point.



30-year commoditization of DBMS and search capability

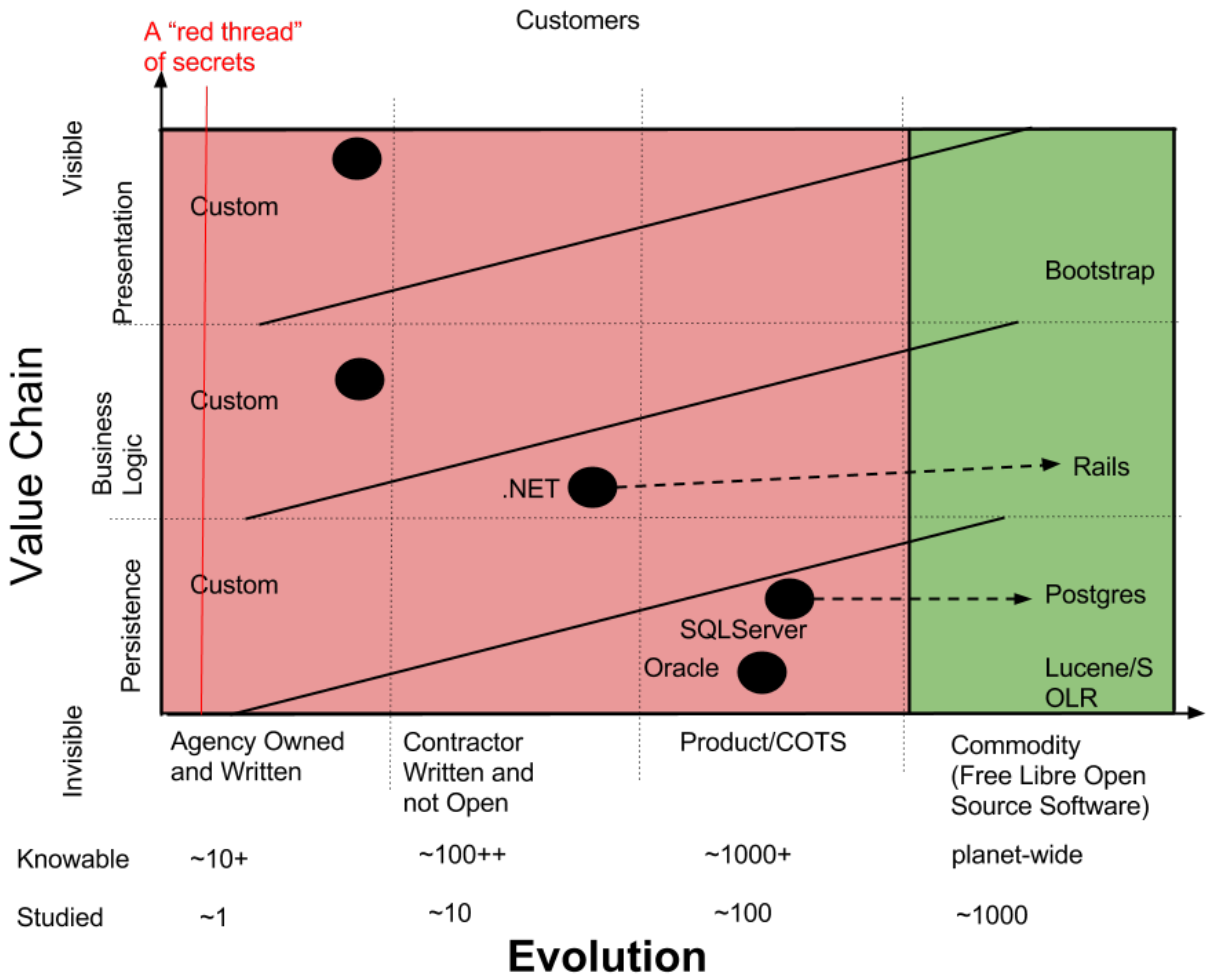
The Point...

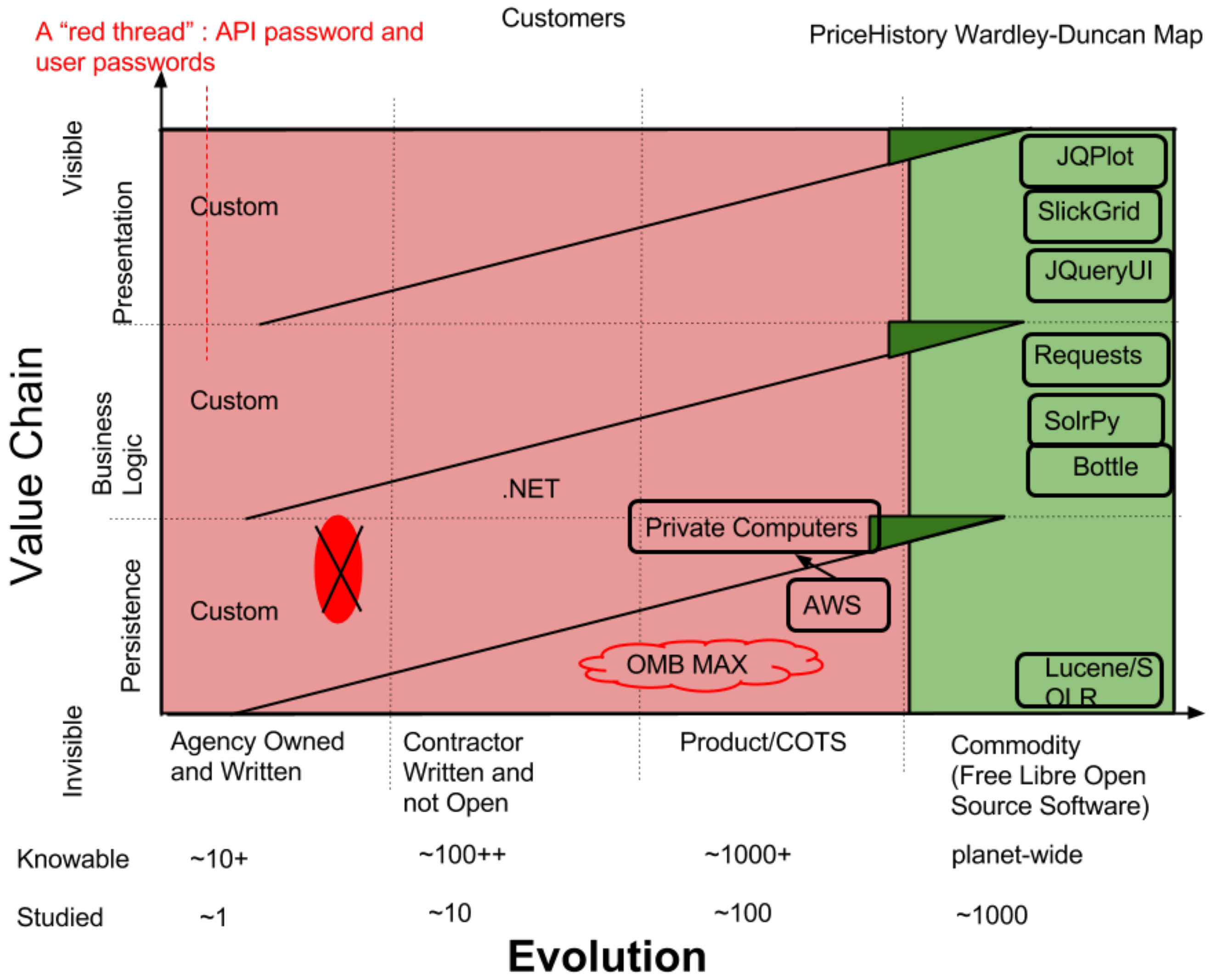
- Don’t pay too much for that which is now a commodity
- Understand that EVERYTHING is in the process of becoming commoditized
- Use that trend to save the taxpayer money


Customers
A "red thread" of secrets
Visible
Presentation
Business Logic
Persistence
Invisible
Value Chain
Custom
Custom
Custom
Drupal/ Wordpress
Bootstrap
Gems/ Modules
Rails
J2EE
.NET
Postgres
MySql
Lucene/SOLR
Oracle
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

How to Use the Diagram...

- This is a three-tier architecture diagram superimposed on a Wardley-Duncan map.
- Place each component in your system on the diagram.
- “Move” by trying to move a component to the right.
- Try to minimize lines of code in the custom area.


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


A "red thread" : API password and user passwords
Customers
PriceHistory Wardley-Duncan Map
Visible
Presentation
Business Logic
Persistence
Invisible
Value Chain
Custom
Custom
Custom
.NET
Private Computers
AWS
OMB MAX
JQPlot
SlickGrid
JQueryUI
Requests
SolrPy
Bottle
Lucene/SOLR
Agency Owned and Written
Contractor Written and not Open
Product/COTS
Commodity (Free Libre Open Source Software)
Knowable
~10+
~100++
~1000+
planet-wide
Studied
~1
~10
~100
~1000
Evolution

Commoditization of Unstructured Document Websites

Application

Rocket Science

Structured Data, Algorithmic Regulations

Unstructured Data, User Entry of Documents

Barely Commoditized $\longrightarrow$ Highly Commoditized

Custom Website

Application Framework (Django, Rails)

Content Management System (Drupal WordPress)

Implementation

The Cloud

... is just other people’s computers.

It is the commoditization of RUNNING computer systems.

Checklist (the Biggest)

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

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.)

Outline

- Commoditization: An Irresistible Force
- Risk Management
- Security

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”.

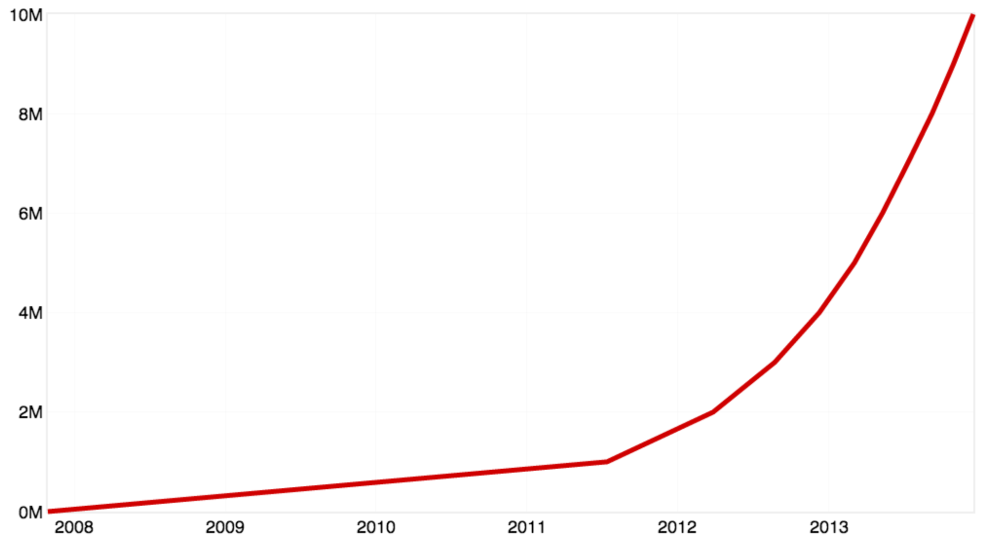
18F

Benjamin Franklin: The First Civic Hacker

- A Uniquely American Fantasy: 10 years from now there will be 100,000 citizens contributing code to Federal Projects
- BUT ONLY UNDER CODE REVIEW!!!
- Less than 0.1% of 18F code has been contributed by external sources.

The Coming Open Source Singularity

- The Unix Way made real: write small, independently recombinable programs.
- Made real by GitHub and other code-sharing sites.



Programming has Changed...

- Open Source Programmers are now fundamentally more productive.
- Every year Open Source Programmers become 20% more productive (subjective).
- I spend most of my time figuring out how to reuse code.
- No one can afford to be left out of this!

“90% of everything is crud.”

-- Sturgeon’s Law.

- There are 10 Million open source projects on github.
- 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.
- But the crud makes good compost.

Architecture trumps Coding

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

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.

Balance risks/benefits

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?

Why Code you must Write Should be Open

- To minimize costs when a different vendor has to work with it, which will happen in a few years.
- You are writing tomorrow’s legacy codebase, and have a responsibility to minimize the burden of replacing it.

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.

Outline

- Commoditization: An Irresistible Force
- Risk Management
- Security

Security Principles

- Risks you can see are better than risks you cannot see.
- No “Security by Obscurity”.
- More eyeballs means decreased risks.
- Assume that your secrets are known and change them frequently.
- Easier to keep small, changeable secrets (a red thread)
- A codebase is the worst possible secret!

18F

Security: Reuse of Existing Projects

- Small, code reviewable utilities by individuals and small communities
- Big, highly supported and widely deployed pillars
- Take the money you save from reuse and put it into penetration and code review testing
- Automated security testing is necessary but insufficient
- Be educated, but not a “shiny object person”

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 and end-of-life of project as inexpensive as possible

Q: What makes Open Source Secure?

A: Enough eyeballs are on it.

You must force sufficient code review--the more independent the better.

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

FISMA-public : a pseudo-category

- NIST defines High, Moderate, and Low…
- But probably should also define “public”…
- Read-only data which cannot be entered by the public…
- Still must avoid defacement.

FISMA

- Try to avoid mixing FISMA-levels in the same system.
- Modularity is your only hope here.
- Try to use inherited controls to make your life easier.

Further Reading

- http://blog.gardeviance.org/2014/02/a-wardley-map.html -- Wardley-Duncan maps by Simon Wardley
- 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
- http://en.wikipedia.org/wiki/Federal_Information_Security_Management_Act_of_2002 -- FISMA
- http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf -- definition for Low, Moderate, High
- https://github.com/blog/1724-10-million-repositories -- GitHub has 10 M repositories.