With Age Comes Wisdom (Hopefully!): *Lessons Learned in 15 Years of Building Software*

Daniel Bryant

@danielbryantuk

BigPictureTech

tl;dr

- This talk is primarily about communication
- Defining and effectively sharing the vision and goals is essential
- Using and agreeing on models reduces friction
- Optimising for feedback and learning is a core skill

@danielbryantuk



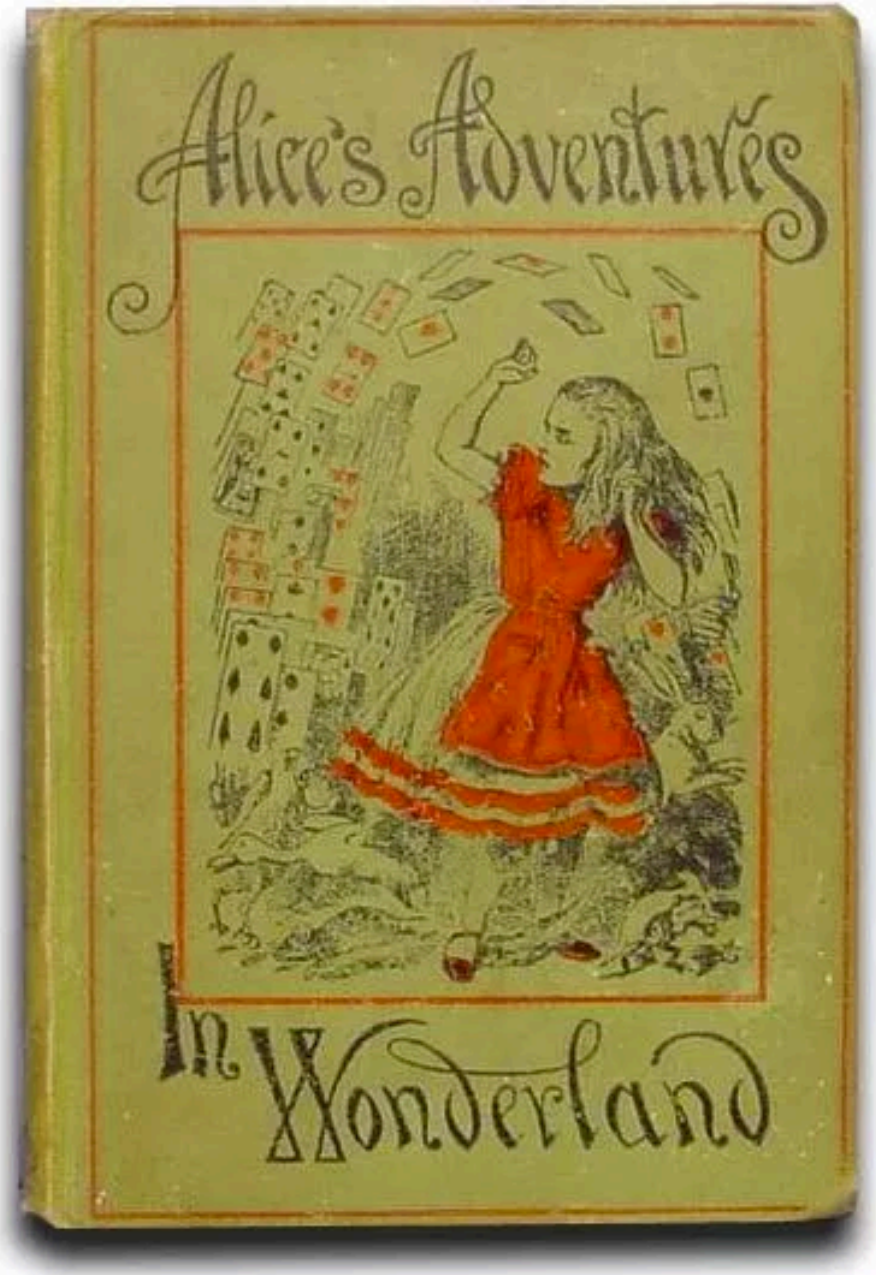
- Tech Consultant, Product Architect at Datawire, and InfoQ Editor
  - Architecture, DevOps, Java, microservices, cloud, containers
  - Continuous Delivery (CI/CD) advocate
  - Leading change through technology and teams

oreil.ly/2E63nCR

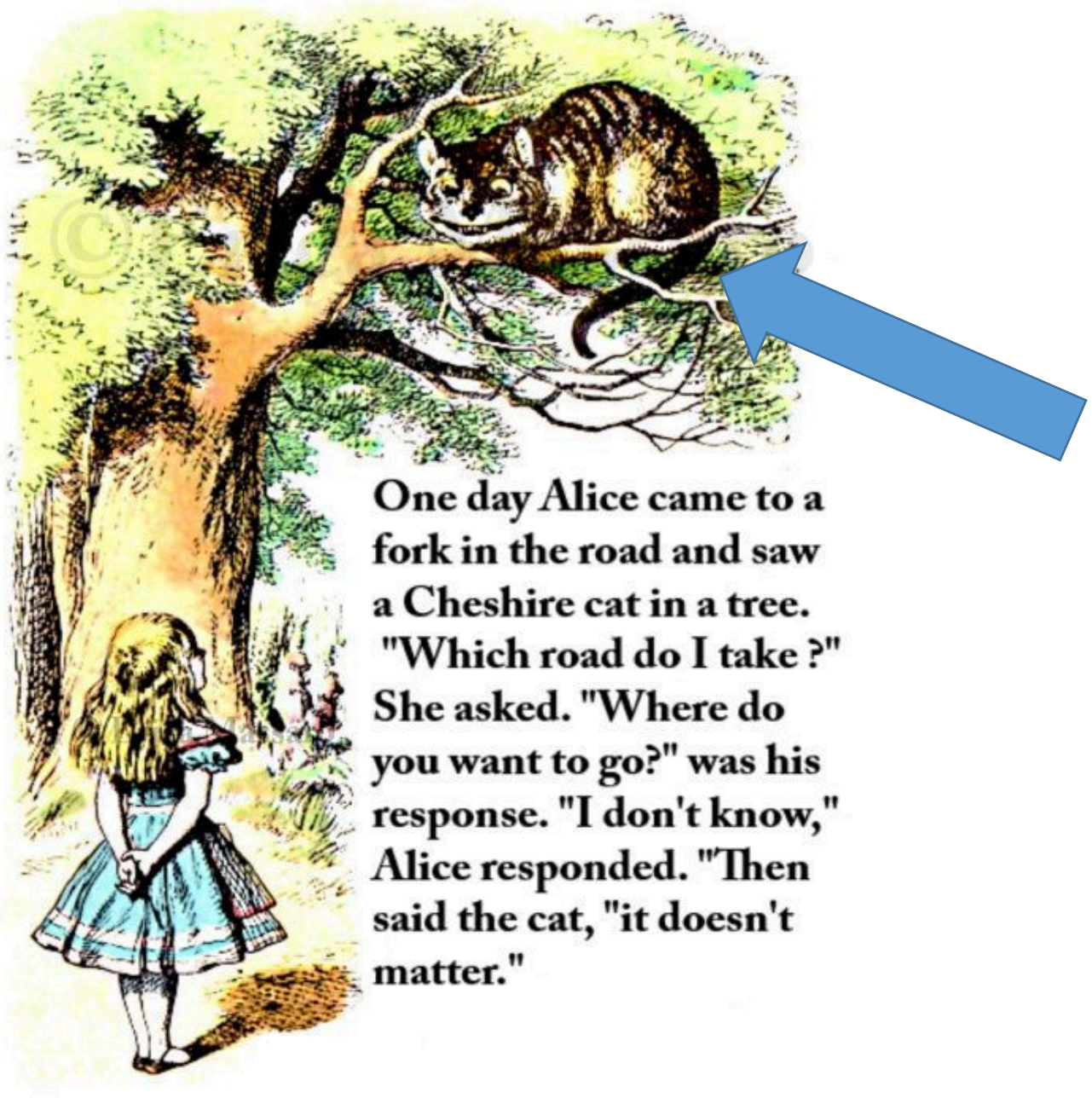


bit.ly/2jWDSF7

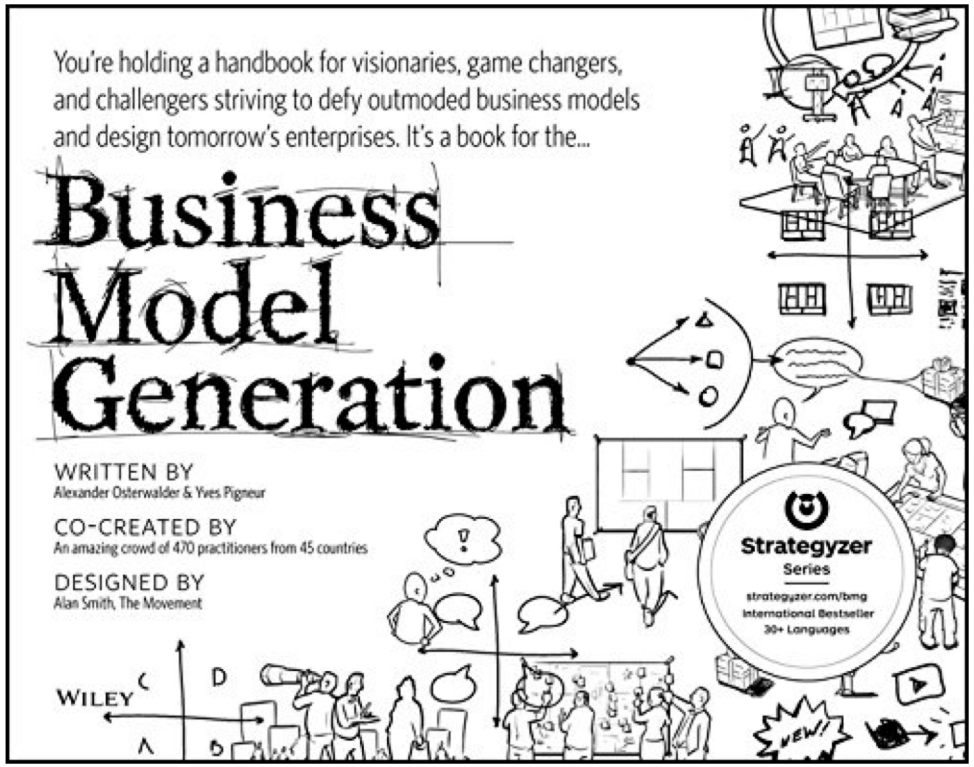
Defining and sharing the vision



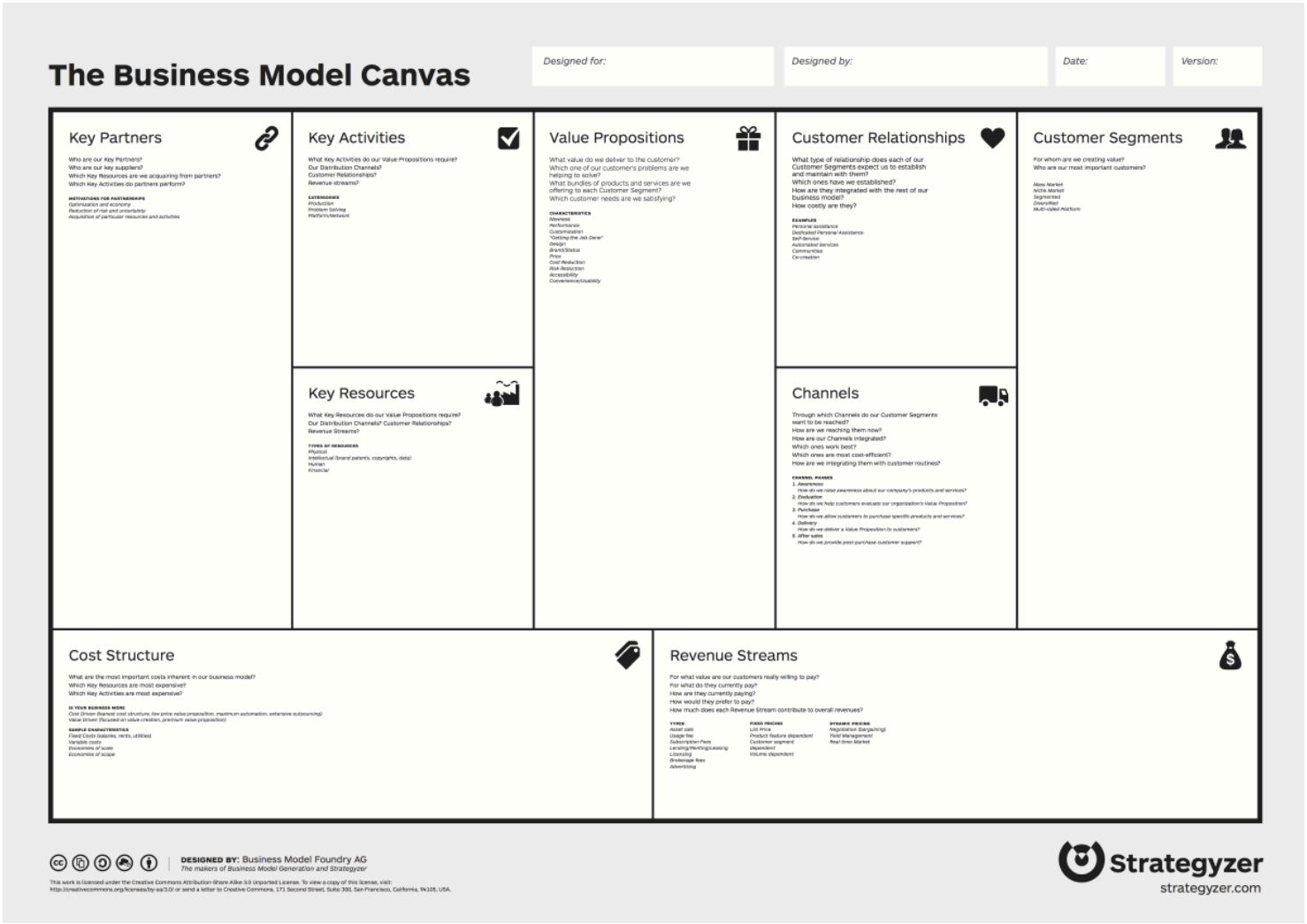
BigPictureTech

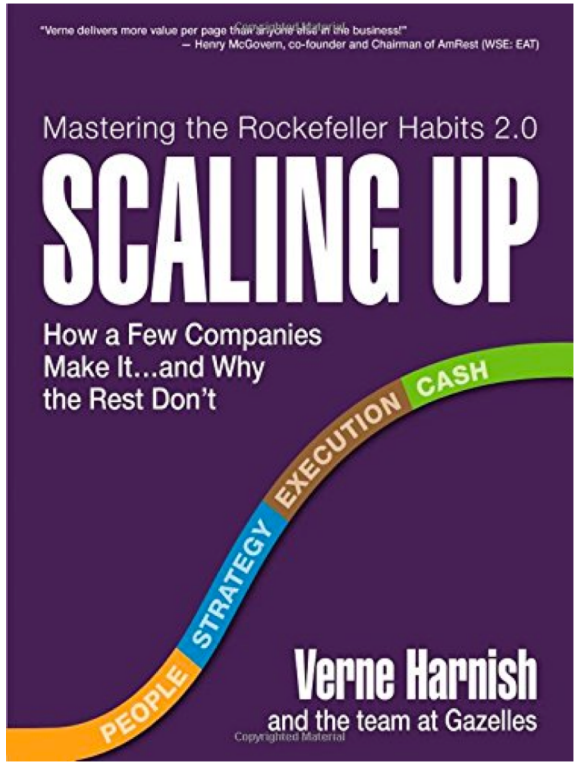


© Erica Massaro
EDMPoetryPhotography

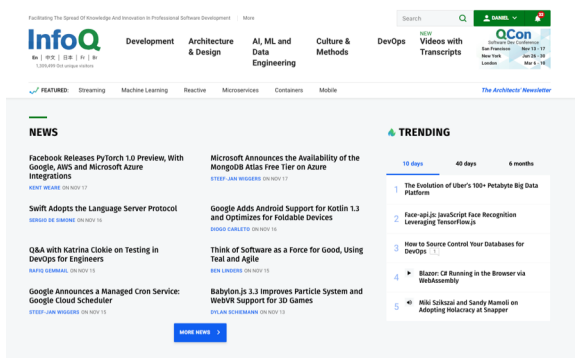


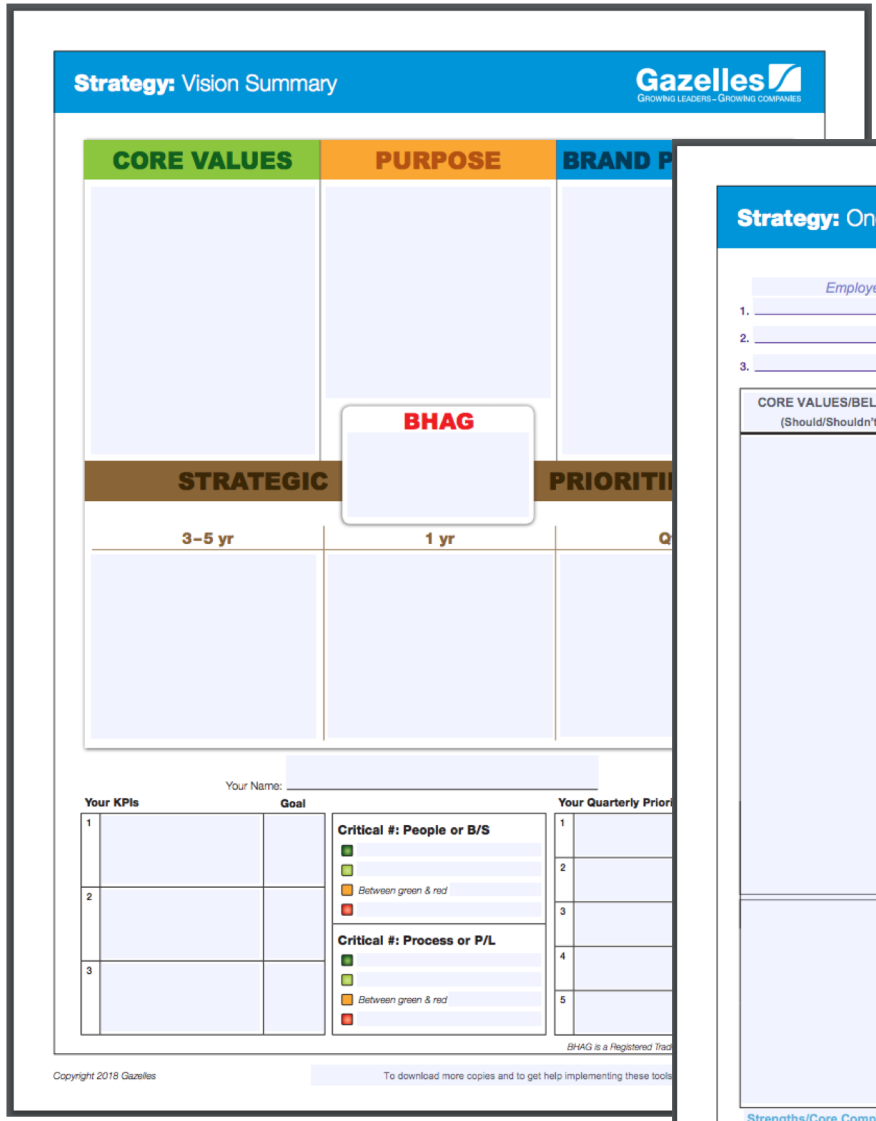
https://strategyzer.com/books/business-model-generation

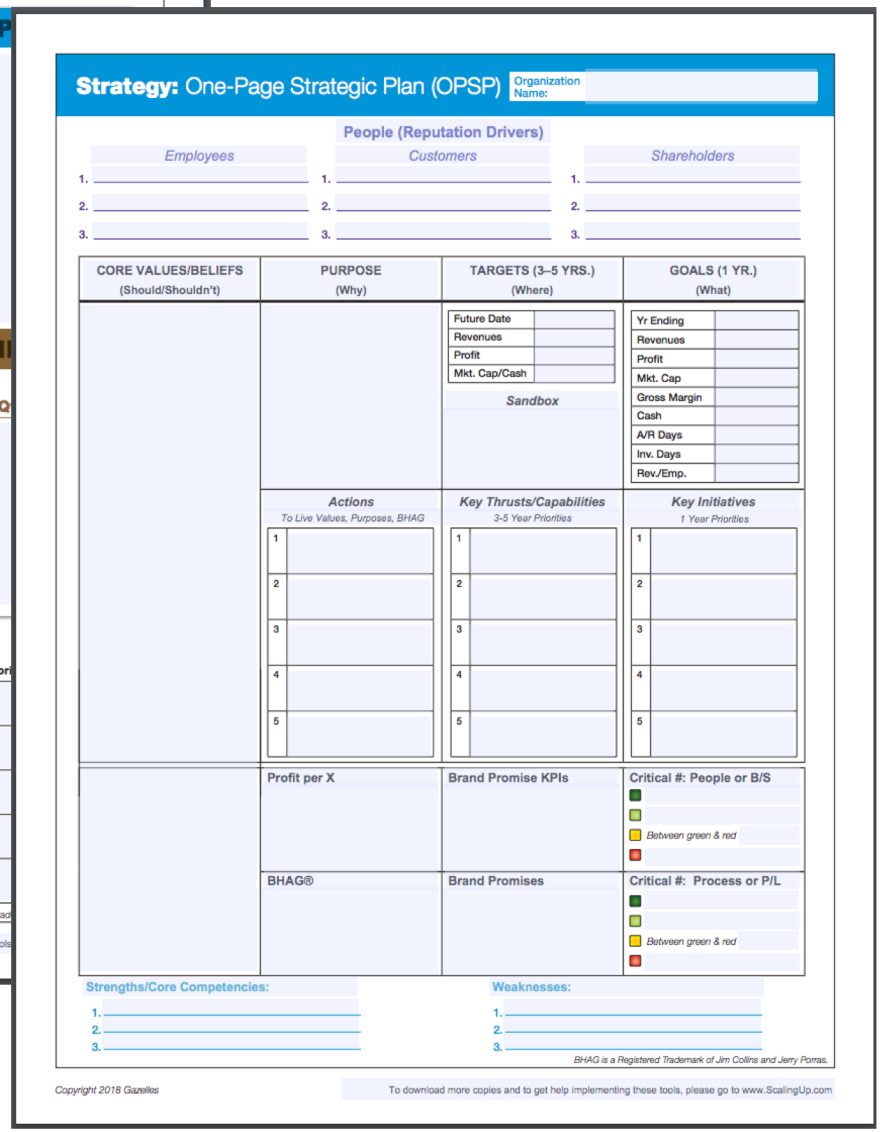




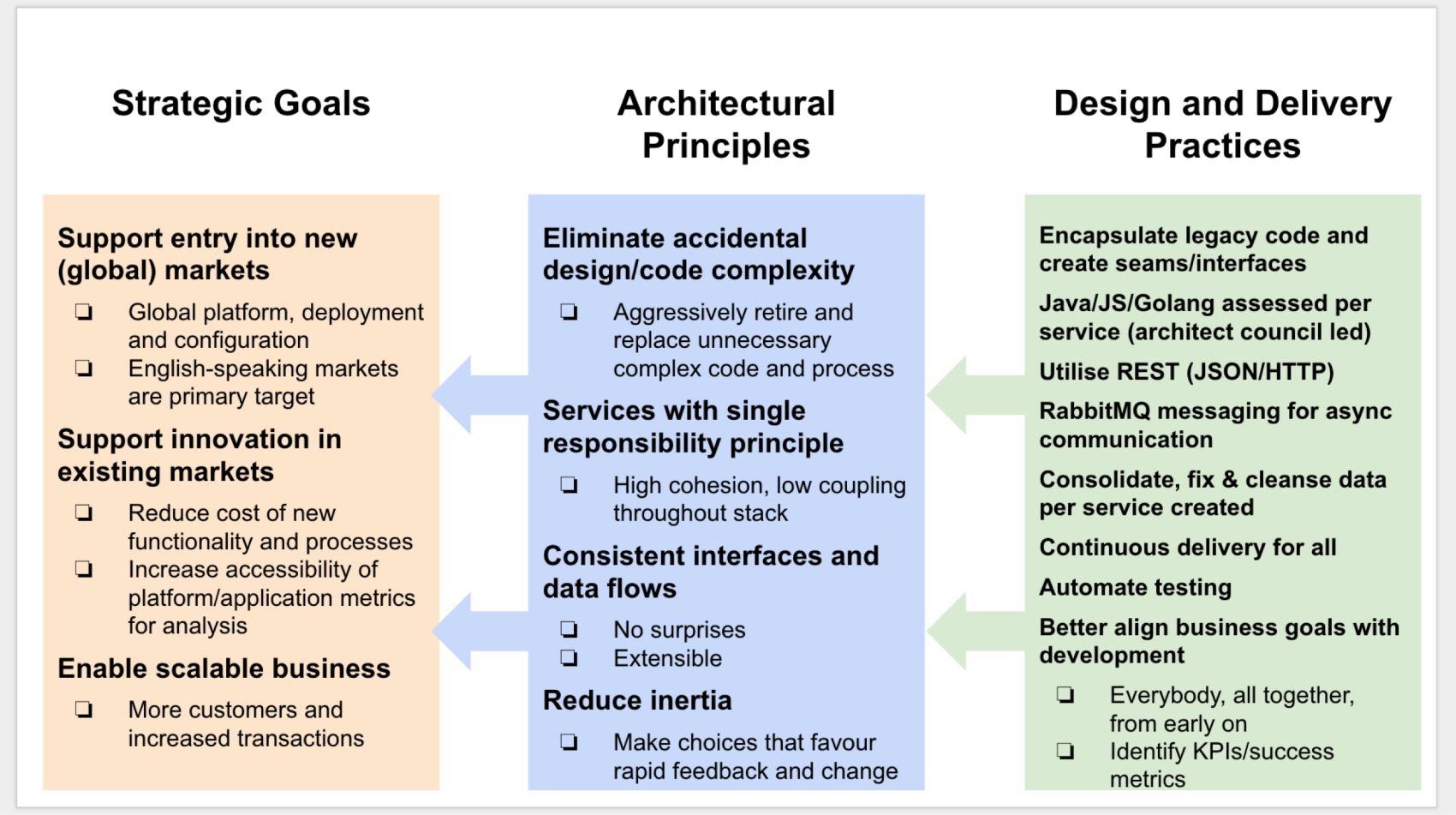
https://gazelles.com/resources/growth-tools



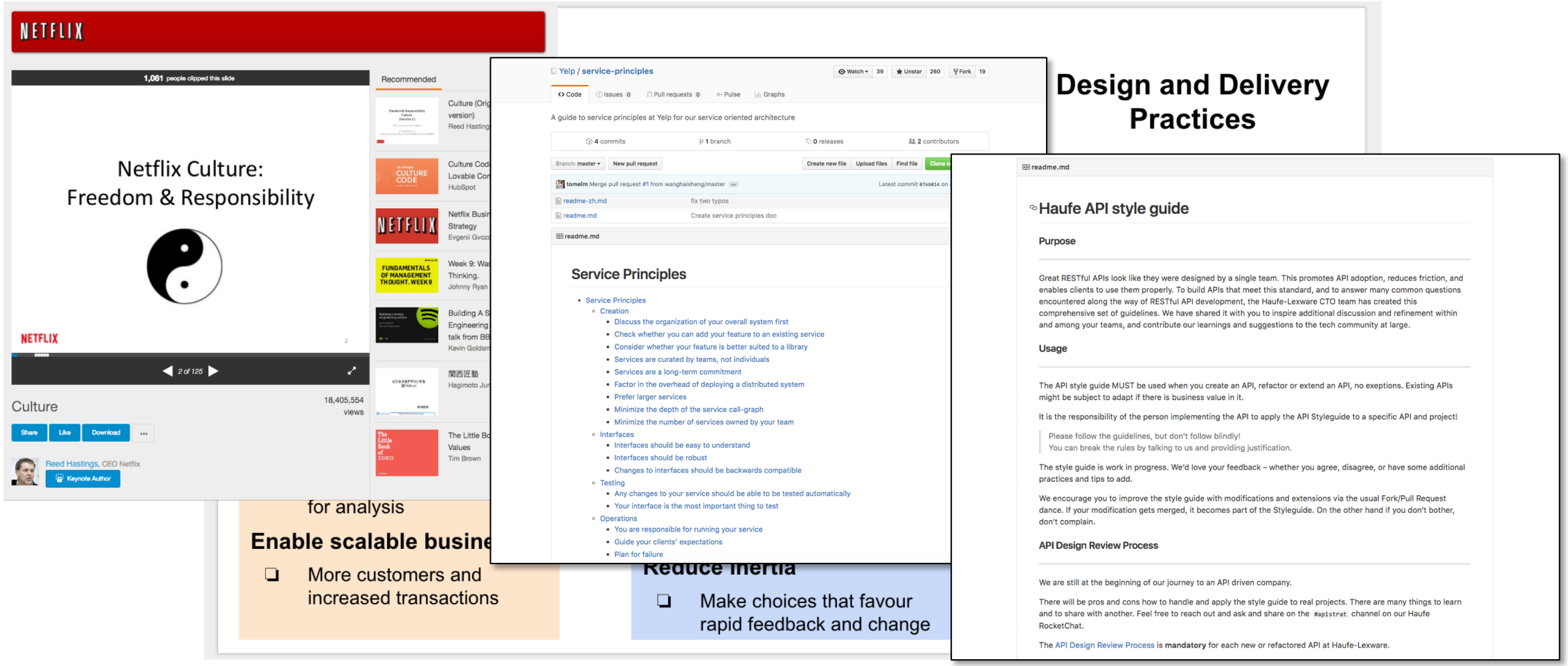




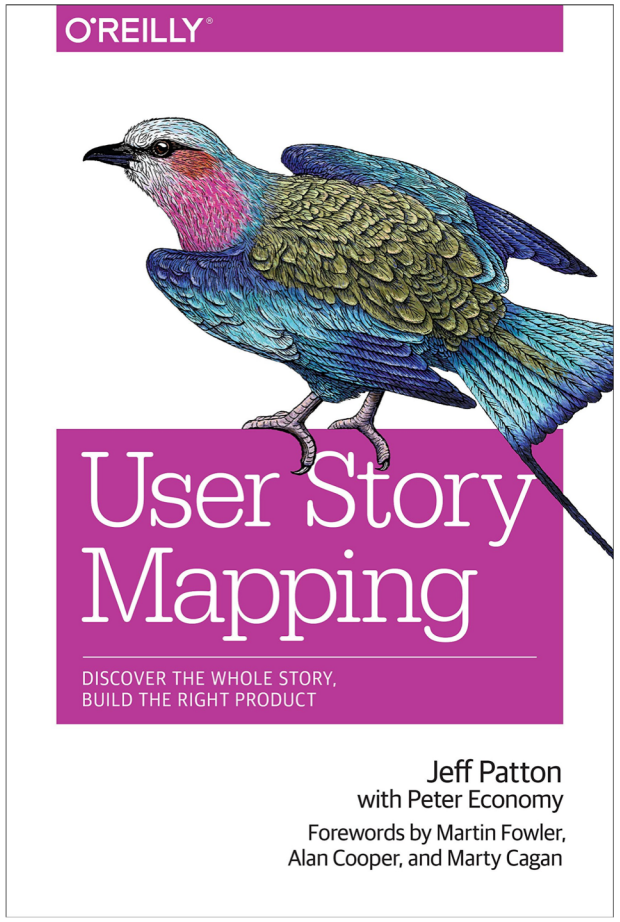
Share the vision (increase empathy)

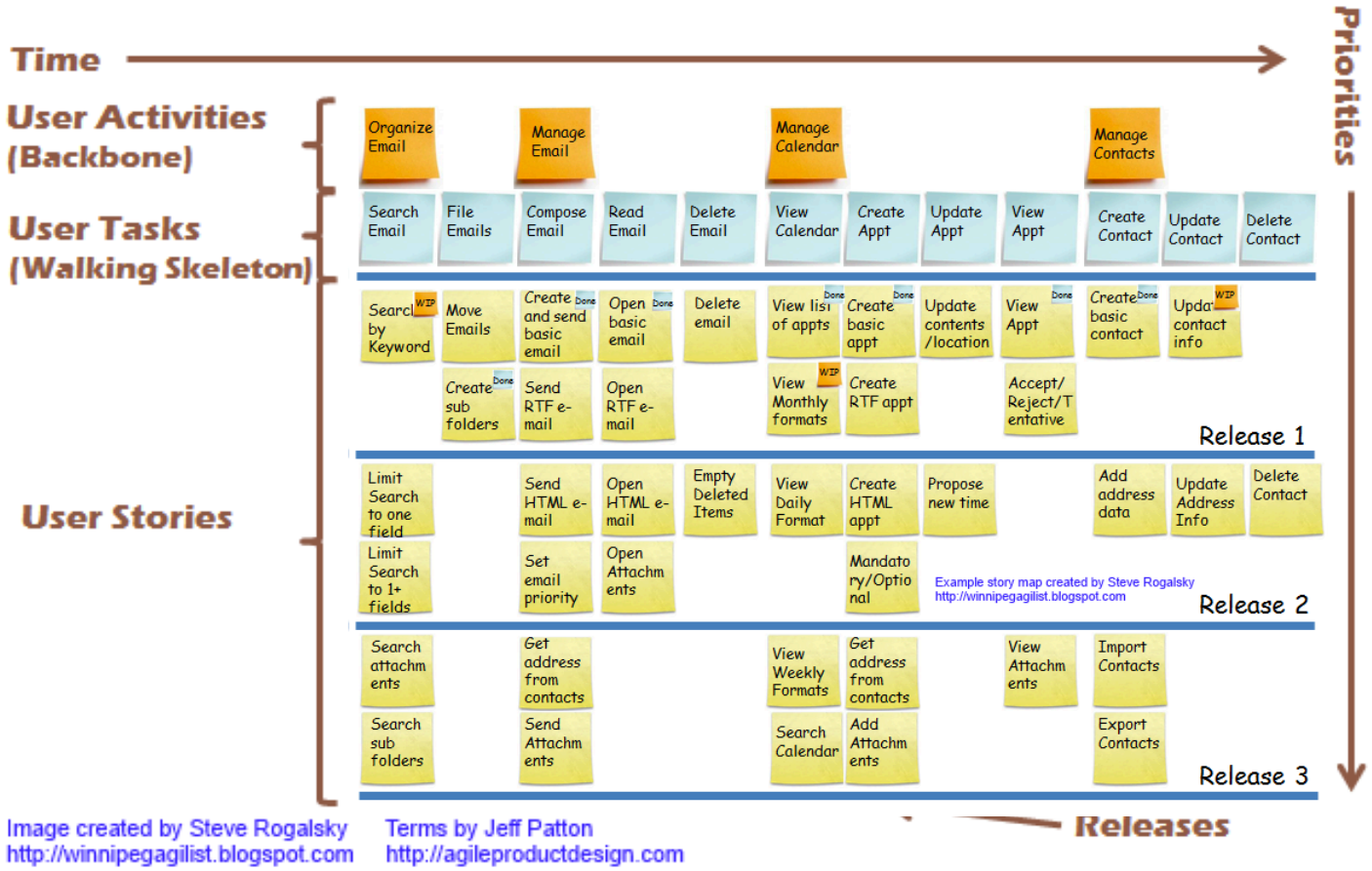


Share the technical vision (increase empathy)



Share the user journey





All models are wrong…

The most challenging (de)serialization problem



MARTINFOWLER.COM

Intro Videos Design Agile Refactoring FAQ About Me All Sections ThoughtWorks

UbiquitousLanguage

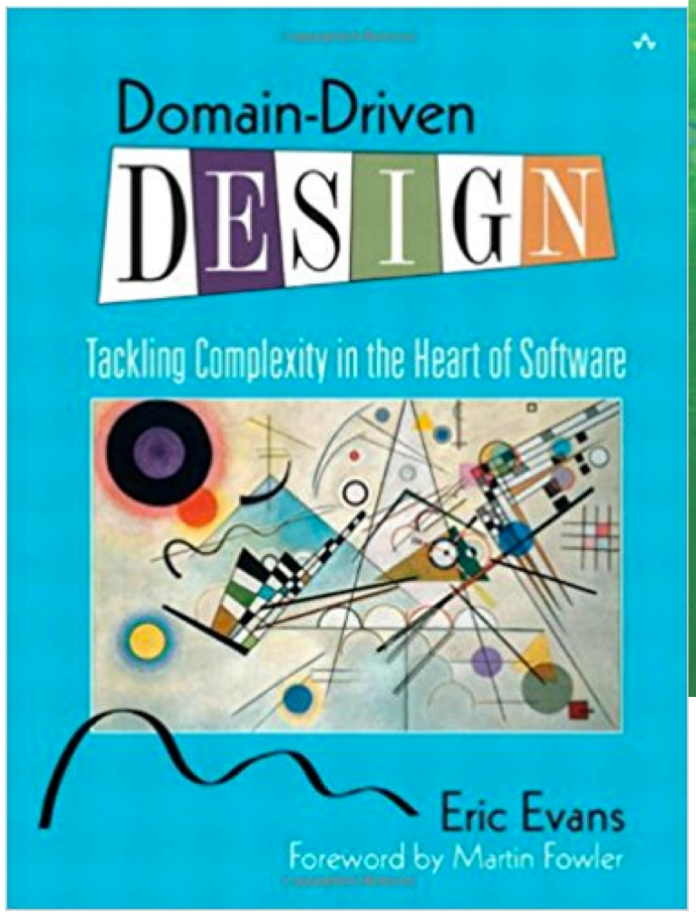


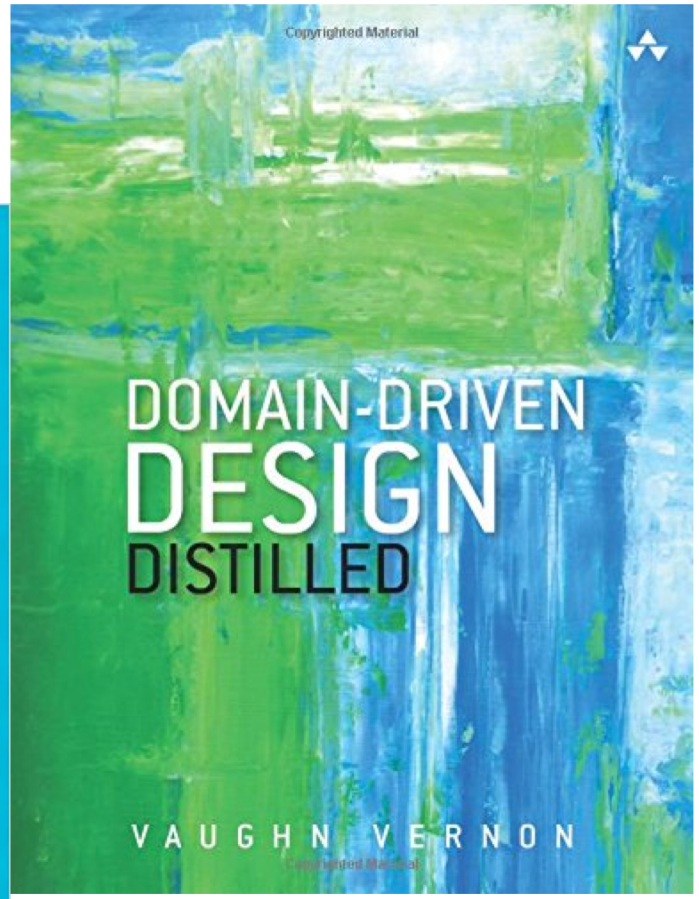
Martin Fowler
31 October 2006

Ubiquitous Language is the term Eric Evans uses in Domain Driven Design for the practice of building up a common, rigorous language between developers and users. This language should be based on the Domain Model used in the software - hence the need for it to be rigorous, since software doesn't cope well with ambiguity.

Evans makes clear that using the ubiquitous language between in conversations with domain experts is an important part of testing it, and hence the domain model. He also stresses that the language (and model) should evolve as the team's understanding of the domain grows.

> *By using the model-based language pervasively and not being satisfied until it flows, we approach a model that is complete and comprehensible, made up of simple elements that combine to express complex ideas.*
>
> ...
>
> *Domain experts should object to terms or structures that are awkward or inadequate to convey domain understanding; developers should watch for ambiguity or inconsistency that will trip up design.*
>
> *-- Eric Evans*





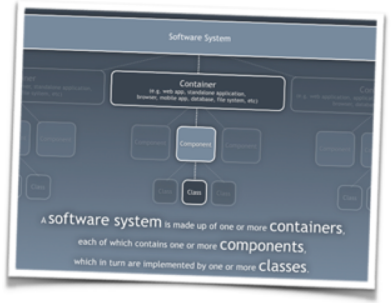
Creating a common (technical) language

Software architecture and the C4 model

coding (the) architecture

1 Agree on a simple set of abstractions that the whole team can use to communicate.

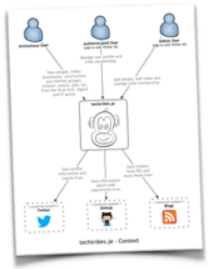
A common set of abstractions is more important than a common notation, but do ensure that your notation (shapes, colours, line styles, acronyms, etc) is understandable. Add a key/legend if in doubt.



2 Draw a number of simple diagrams at different levels of abstraction.
C4: context, containers, components, classes

Naming
If naming is one of the hardest things in software development, resist the temptation to have a diagram full of boxes that only contain names!

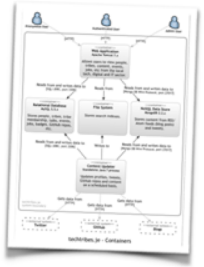
Adding a brief statement of responsibilities to containers and components is a simple way to remove ambiguity. It provides a nice "at a glance" view too.



1. System context diagram

A context diagram can be a useful starting point for diagramming and documenting a software system, allowing you to step back and look at the big picture. Draw a simple block diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interfaces with.

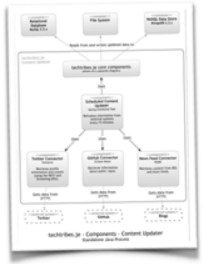
Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.



2. Container diagram

Once you understand how your system fits in to the overall IT environment with a context diagram, a really useful next step can be to illustrate the high-level technology choices with a containers diagram. By "container" I mean something like a web application, desktop application, mobile app, database, file system, etc. Essentially, what I call a container is anything that can host code or data.

A container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focussed diagram that is useful for software developers and support/operations staff alike.



3. Component diagram(s)

Following on from a container diagram showing the high-level technology decisions, I'll then start to zoom in and decompose each container further. However you decompose your system is up to you, but I tend to identify the major logical components and their interactions. This is about partitioning the functionality implemented by a software system into a number of distinct components, services, subsystems, layers, workflows, etc.

The components diagram shows how a container is divided into components, what each of those components are, their responsibilities and the technology/implementation details.

How would you code it?
Ask yourself this question if you can't figure out how to draw something.

4. Class diagram(s)

This is optional, but I sometimes draw one or more UML class diagrams if I want to illustrate specific component implementation details.

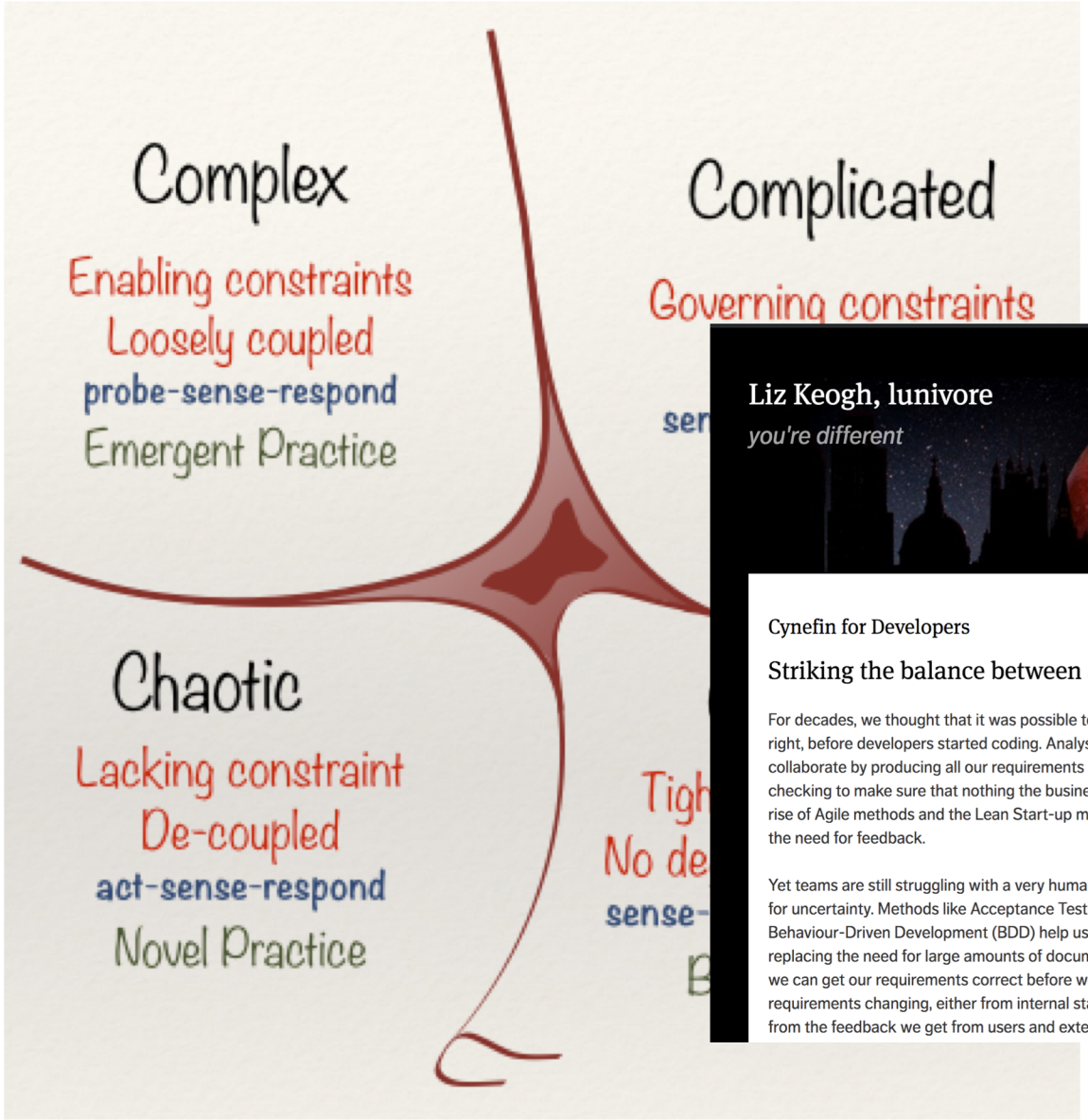


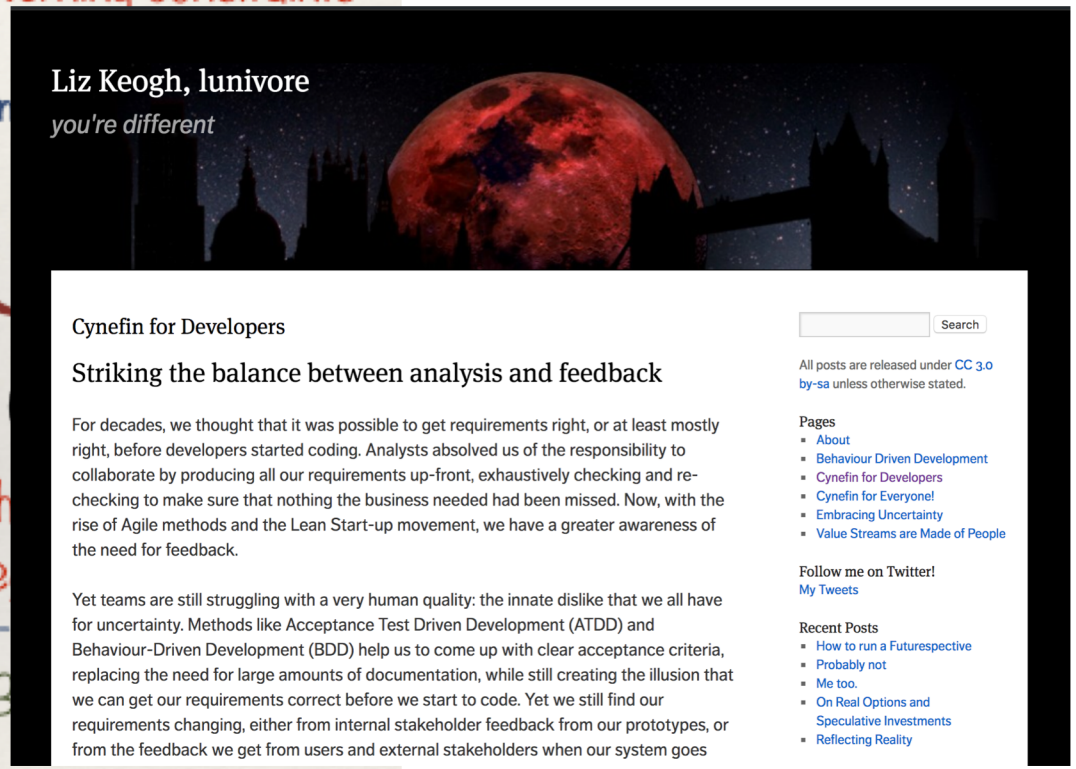
More information about software architecture and the C4 model can be found in "Software Architecture for Developers", available as an ebook from leanpub.com

@simonbrown | www.codingthearchitecture.com

https://c4model.com/

Creating a problem definition language





https://lizkeogh.com/cynefin-for-developers/

https://en.wikipedia.org/wiki/Cynefin_framework

“Some decisions are consequential and irreversible or nearly irreversible – one-way doors – and these decisions must be made methodically, carefully, slowly, with great deliberation and consultation. […]

*We can call these **Type 1 decisions.** “*

- Jeff Bezos, 2017 Letter to Shareholders

*“But most decisions aren’t like that – they are changeable, reversible – they’re two-way doors. If you’ve made a suboptimal **Type 2 decision**, you don’t have to live with the consequences for that long. […]*

Type 2 decisions can and should be made quickly by high judgment individuals or small groups.”

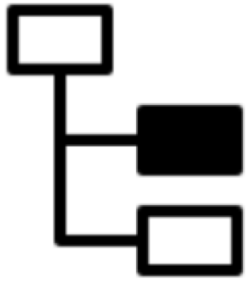
- Jeff Bezos, 2017 Letter to Shareholders

Netflix Play API

Why we built an Ev… Architecture

Suudhan Rangarajan (@suudhan)
Senior Software Engineer

Three Type 1 Decisions to Consider



Appropriate Coupling



Synchronous & Asynchronous



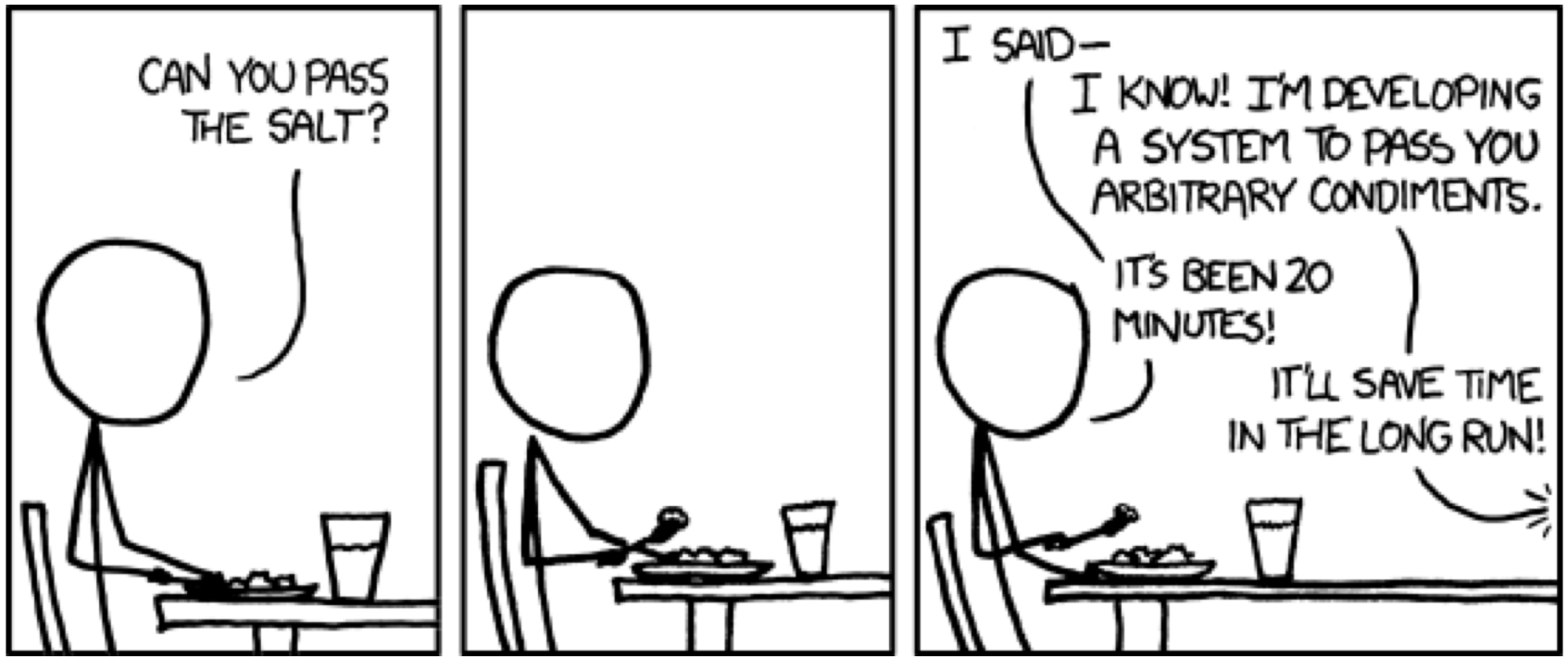
Data Architecture

NETFLIX

https://qconsf.com/system/files/presentation-slides/qcon_netflix_play_api.pdf

Evaluation


BGN 111
© SWNS



https://xkcd.com/974/

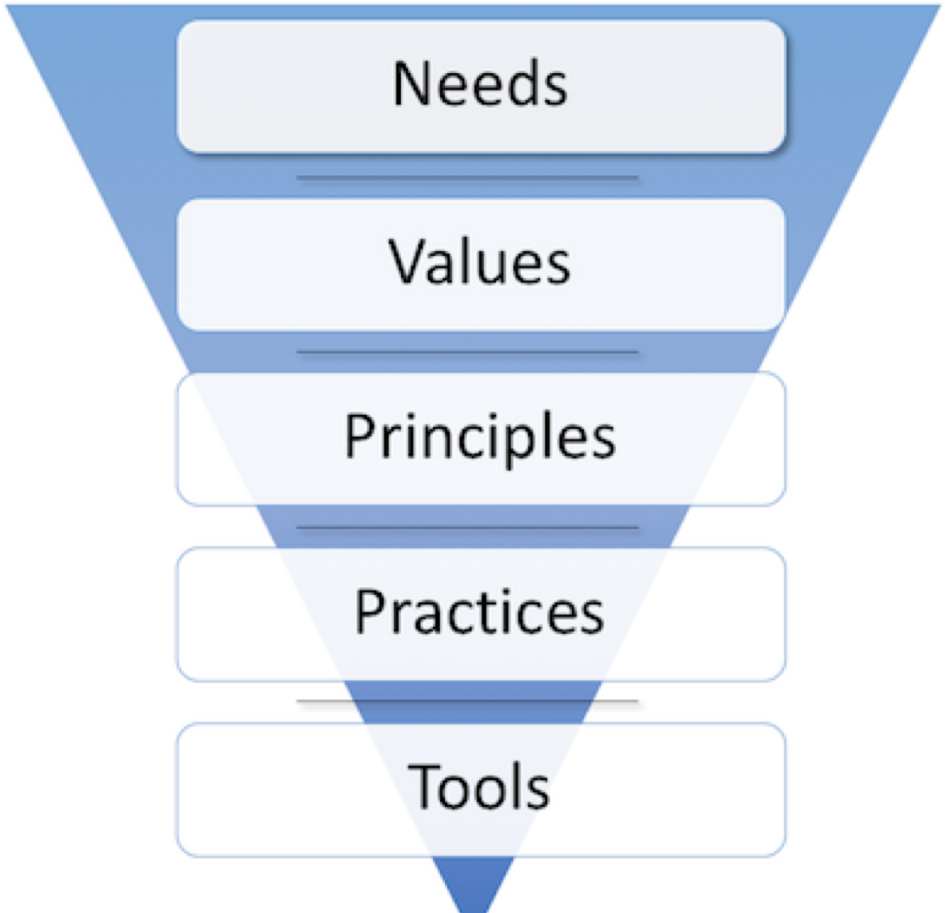
Evaluation - first steps

- Frame the problem, and the general solution
- Offer at least two alternatives
- State the recommendation
  - Must include benefits and drawbacks (“tradeoffs”)

- Cultivate empathy (Dev/Ops - Ops/Dev)

The Spine Model

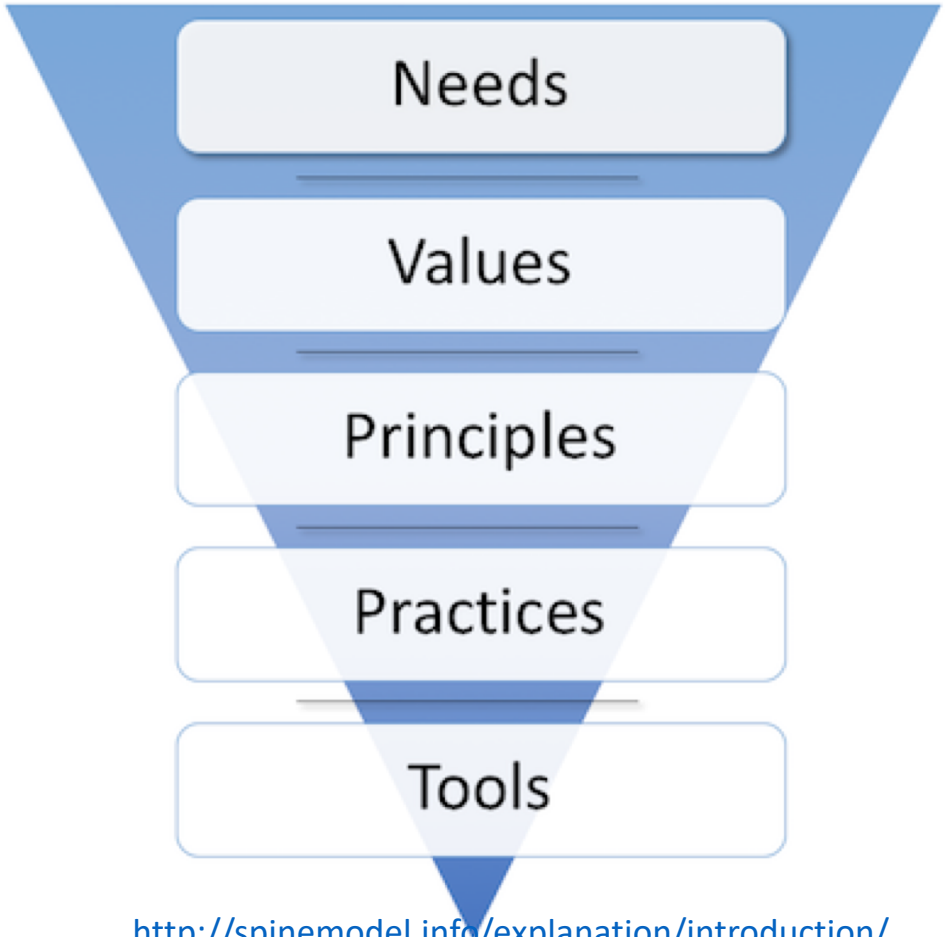
- Effective conversations make for effective collaboration
- It's a TOOL Problem
- People get stuck in a dilemma where equally plausible options are available
- “Going up the Spine” breaks deadlock



http://spinemodel.info/explanation/introduction/

Determine the need for the tool

- PRACTICES before Tools
  - Decide on the Practices that the tools are there to support
  - We do _____ to create value
- PRINCIPLES before Practices
  - Decide on the Principles to measure those Practices against.
  - We leverage _____ to change the system
- VALUES before Principles
  - Make as explicit as possible the Values at play in the system.
  - We optimise for _____
- NEEDS before Values
  - It all starts at Needs. Why does this system exist in the first place?
  - We are here to satisfy _____



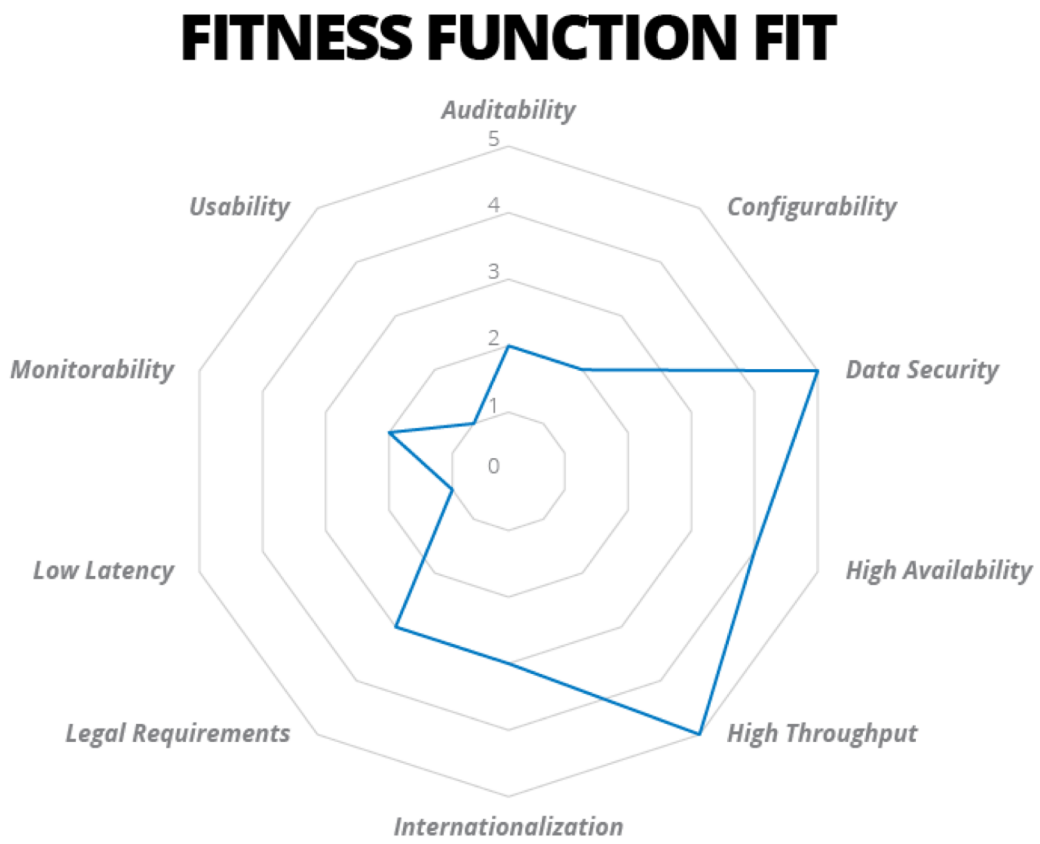
http://spinemodel.info/explanation/introduction/

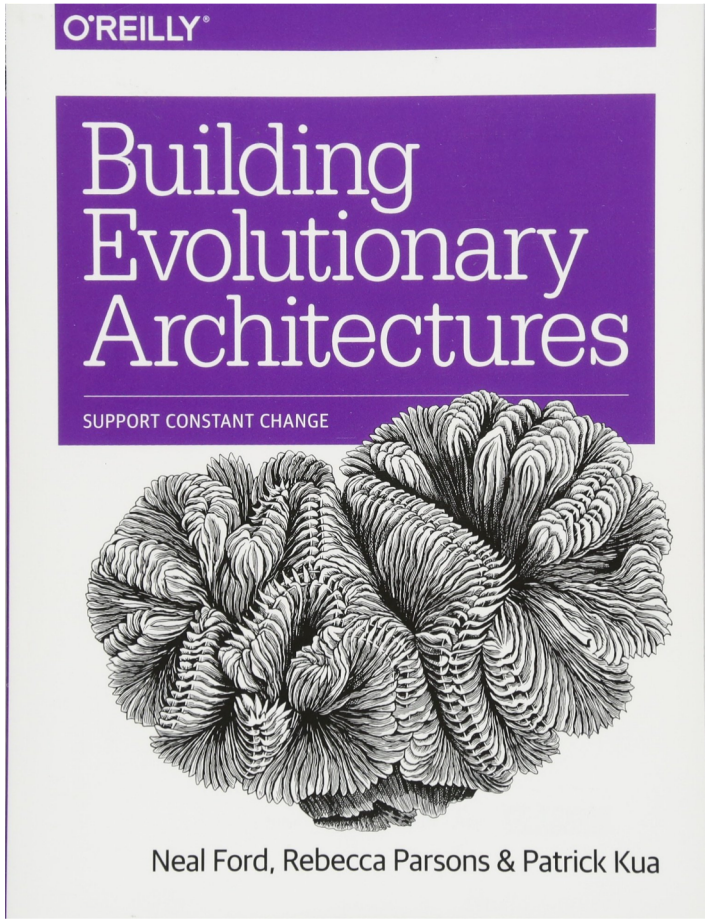
Making (type 1) tooling decisions

Matt Raible's Comparison Framework

Criteria	Struts 2	Spring MVC	Wicket	JSF 2	Tapestry	Stripes	GWT	Grails	Rails	Flex	Vaadin	Lift	Play
Developer Productivity	0.50	0.50	0.50	0.50	1.00	0.50	1.00	1.00	1.00	0.00	1.00	0.50	1.00
Developer Perception	0.50	1.00	1.00	0.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Learning Curve	1.00	1.00	0.50	0.50	0.50	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00
Project Health	0.50	1.00	1.00	1.00	0.50	0.50	1.00	1.00	1.00	0.50	1.00	1.00	1.00
Developer Availability	0.50	1.00	0.50	1.00	1.00	0.50	1.00	0.50	1.00	1.00	0.50	0.00	0.50
Job Trends	1.00	1.00	0.50	1.00	0.50	0.00	1.00	0.50	1.00	1.00	0.00	0.00	0.50
Templating	1.00	1.00	1.00	0.50	1.00	1.00	0.50	1.00	1.00	0.50	0.50	0.50	0.50
Components	0.00	0.00	1.00	1.00	1.00	0.00	0.50	0.50	0.50	1.00	1.00	0.00	0.00
Ajax	0.50	1.00	0.50	0.50	0.50	0.50	1.00	0.50	0.50	0.50	1.00	1.00	0.50
Plugins or Add-Ons	0.50	0.00	1.00	1.00	0.50	0.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00
Scalability	1.00	1.00	0.50	0.50	0.50	1.00	1.00	0.50	0.50	0.50	0.50	1.00	1.00
Testing	1.00	1.00	0.50	0.50	1.00	1.00	0.50	1.00	1.00	0.00	0.50	0.50	1.00
i18n and l10n	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	0.50	0.50	1.00	1.00	1.00
Validation	1.00	1.00	1.00	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	0.50
Multi-language Support (Groovy / Scala)	0.50	0.50	1.00	1.00	1.00	1.00	0.00	1.00	0.00	0.00	1.00	0.00	0.50
Quality of Documentation/Tutorials	0.50	1.00	0.50	0.50	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Books Published	1.00	1.00	0.50	1.00	0.50	0.50	1.00	1.00	1.00	1.00	0.50	0.50	0.00
REST Support (client and server)	0.50	1.00	0.50	0.00	0.50	0.50	0.50	1.00	1.00	0.50	0.50	0.50	0.50
Mobile / iPhone Support	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00	1.00	1.00
Degree of Risk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	0.50	0.50
Totals	14.5	17	15	13.5	15	14	17	17.5	17	13.5	15.5	11.5	14

Fitness functions





A couple of (massively generalized) recommendations

Always assume good intentions

even if you don't understand people's behavior

They may simply have a different style.



https://www.pinterest.com/pin/143552306855690834/

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file

Hanlon's razor

From Wikipedia, the free encyclopedia

Hanlon's razor is an aphorism expressed in various ways, including:

> Never attribute to malice that which is adequately explained by stupidity.[1]

It suggests a way of eliminating unlikely explanations ("attributions") for human behavior and i
this kind are known as philosophical razors. It is an eponymous law, probably named after a F

Inspired by Occam's razor,[2] the aphorism was popularized in this form and under this name
computer programmer slang.[3][1] In 1990, it appeared in the *Jargon File* described as a "'murp
Razor".[4] Later that same year, the *Jargon File* editors noted lack of knowledge about the terr
a similar epigram by William James.[1][5] In 1996, the *Jargon File* entry on Hanlon's Razor not
quotation in Robert A. Heinlein's short story "Logic of Empire" (1941), with speculation that Ha
of "Heinlein's Razor".[6] (The character "Doc" in Heinlein's story described the "devil theory" fa
attributed conditions to villainy that simply result from stupidity.")[7]

In 2001, Quentin Stafford-Fraser published two blog entries citing e-mails from Joseph E. Bigl
originally came from Robert J. Hanlon of Scranton, Pennsylvania, as a submission (credited ir
various jokes related to Murphy's law published in Arthur Bloch's *Murphy's Law Book Two: Mo
Wrong!* (1980).[10] Subsequently, in 2002, the *Jargon File* entry noted the same.[11]

M Upgrade

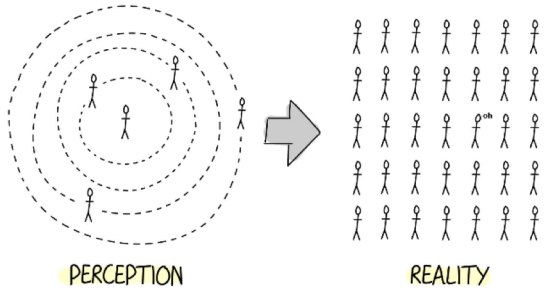
Matthew Cook Follow
Enabling people to reach their potential through positivity, creativity and the occasional crayon.
Apr 16, 2017 · 3 min read

Hanlon's Razor (and why people are nicer than you think)

There is a line in Robert A. Heinlein's short story '*Logic of Empire*' which beautifully reveals the misunderstanding behind most of our social grievances. As one character laments mankind's immorality, another explains:

> *"I would say that you have fallen into the commonest fallacy of all in dealing with social and economic subjects. […] You have attributed conditions to villainy that simply result from stupidity."*

The error is a very human one. We are inherently egocentric. Our understanding of the universe is shaped through our personal experience with it. Life's events unfold for us, not around us. The problem is that we assume we play as prominent a role in everyone else's story. When someone is rude, it is because they are annoyed with us. When someone is upset, it must be something we've done. We perceive a negative response as an act of malice towards us, rather than a result of factors entirely unrelated to our existence. In fact, we perceive a *negative* response when it was most likely neutral.



https://medium.com/@matthewcook/hanlons-razor-f7fa63e70e9a

Dan McKinley

Math, Programming, and Minority Reports

Dan McKinley is a cofounder of Skyliner in Los Angeles.

Previously he worked at Stripe and spent nearly 7 years building Etsy.

Here's Dan's resume and a short bio.

Ask me Anything

Follow @mcfunley

ELSEWHERE

Code
Talks
Photos

EVEN MORE CONTENT



Choose Boring Technology

March 30th, 2015

Probably the single best thing to happen to me in my career was having had Kellan placed in charge of me. I stuck around long enough to see Kellan's technical decisionmaking start to bear fruit. I learned a great deal *from* this, but I also learned a great deal as a *result* of this. I would not have been free to become the engineer that wrote Data Driven Products Now! if Kellan had not been there to so thoroughly stick the landing on technology choices.



Being inspirational as always.

In the year since leaving Etsy, I've resurrected my ability to care about technology. And my thoughts have crystallized to the point where I can write them down coherently. What follows is a distillation of the Kellan gestalt, which will hopefully serve to horrify him only slightly.

Embrace Boredom.

Let's say every company gets about three innovation tokens. You can spend these however you want, but the supply is fixed for a long while. You might get a few more *after* you achieve a certain level of stability and maturity, but the general tendency is to overestimate the contents of your wallet. Clearly this model is approximate, but I think it helps.

If you choose to write your website in NodeJS, you just spent one of your innovation tokens. If you choose to use MongoDB, you just spent one of your innovation tokens. If you choose to use service discovery tech that's existed for a year or less, you just spent one of your innovation tokens. If you choose to write your own database, oh god, you're in trouble.

http://mcfunley.com/choose-boring-technology

Continuous Delivery



Tech

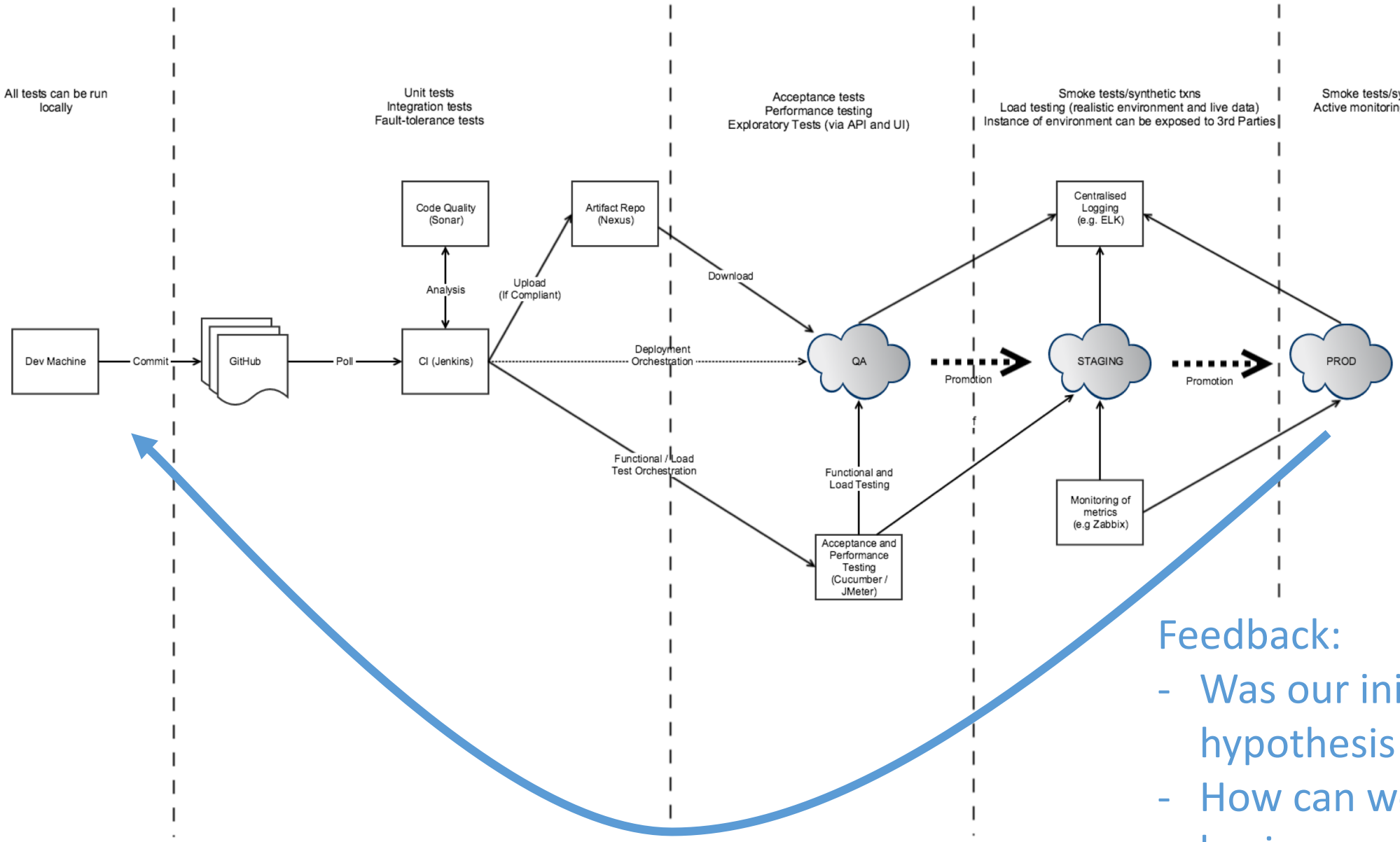


Velocity (with stability) is key to business success

"Continuous delivery is achieved when stability and speed can satisfy business demand.

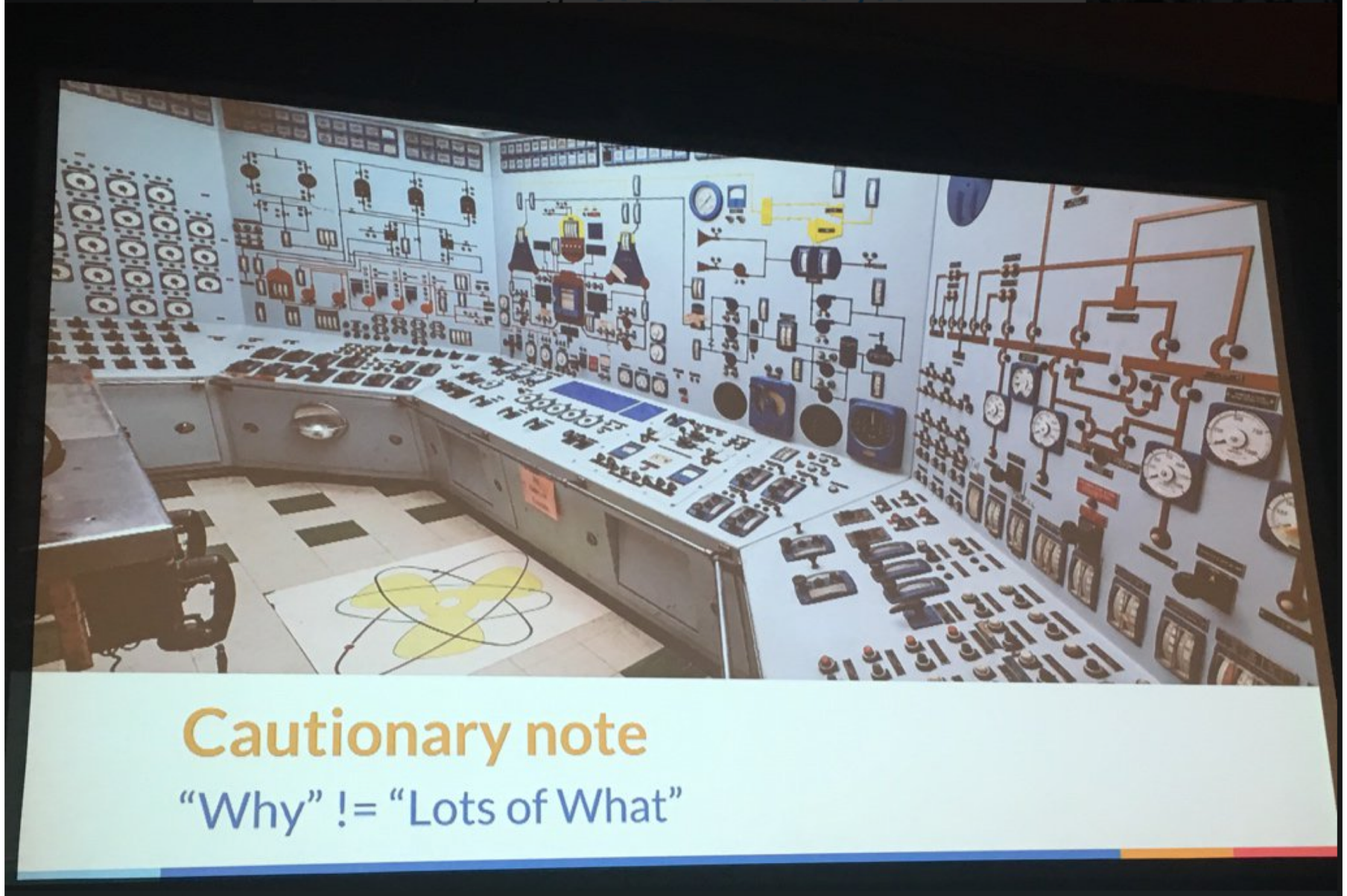
Discontinuous delivery occurs when stability and speed are insufficient."

- Steve Smith (@SteveSmithCD)



Feedback:

- Was our initial hypothesis proven?
- How can we improve business, architecture and ops?

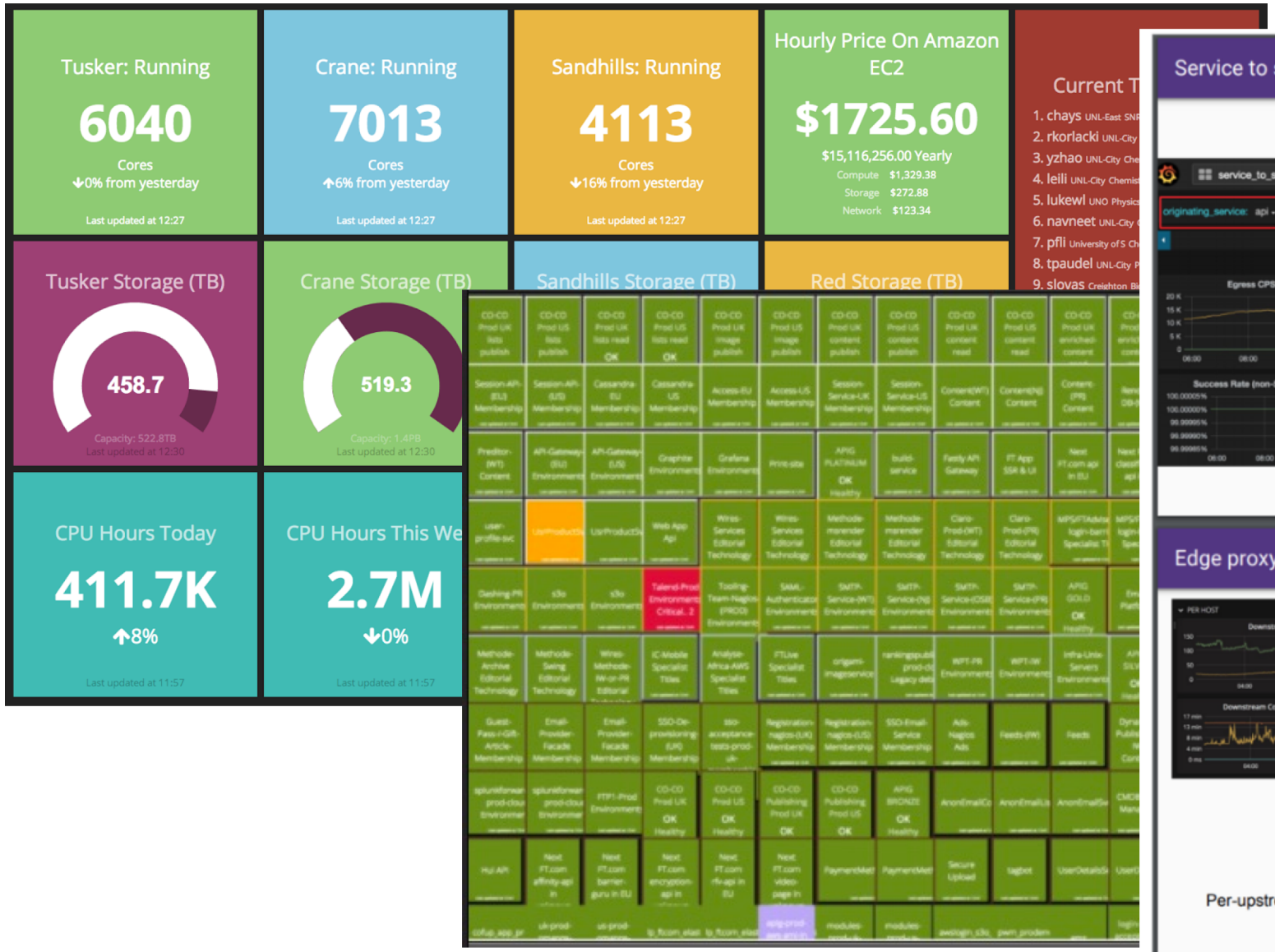


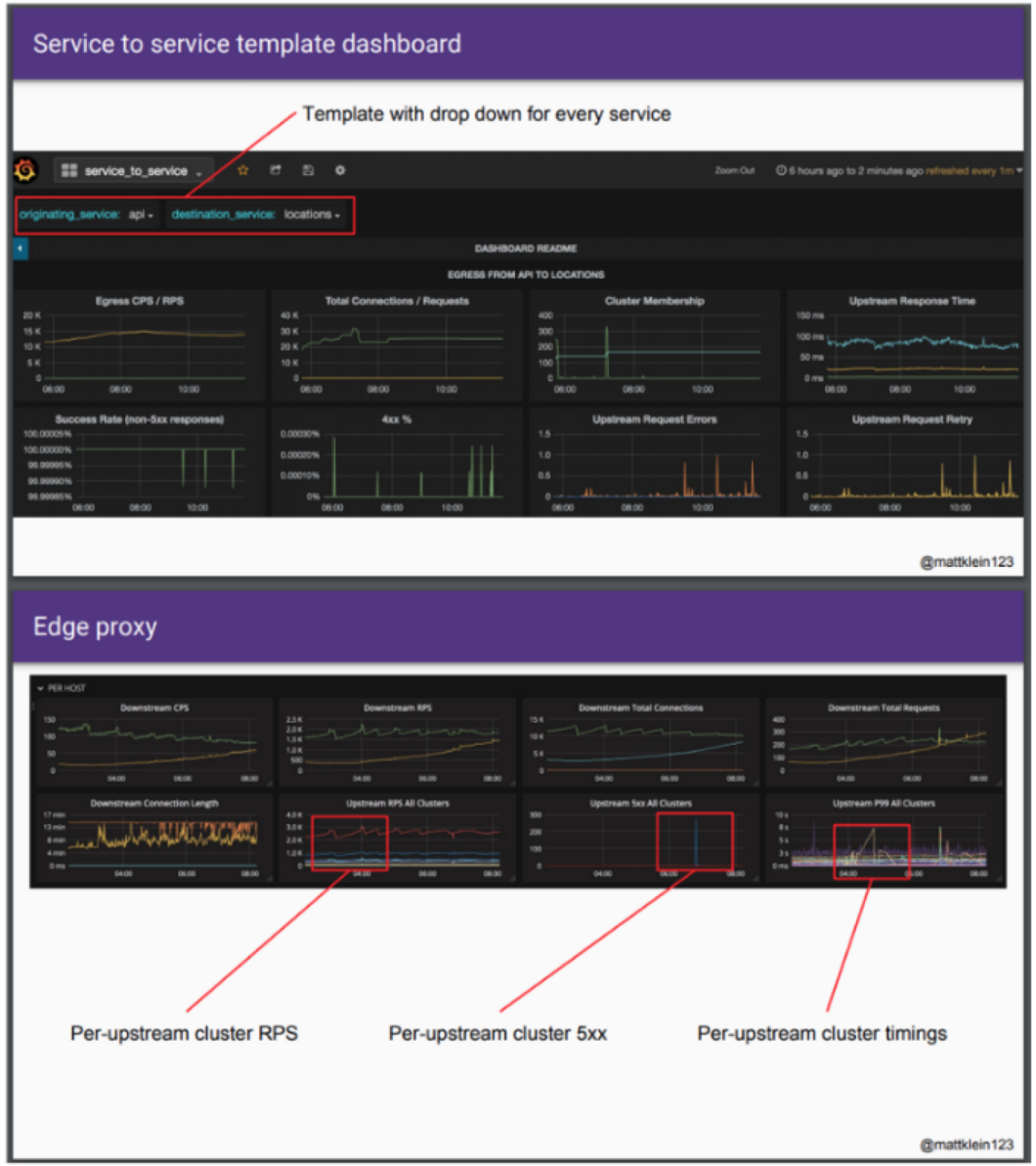
Bridget Kromhout @bridgetkromhout · 21 Jun 2017

Watch out for the nuclear reactor dashboard problem. Having lots of graphs isn't going to solve everything. @el_bhs #velocityconf

1 10 14

gPictureTech





https://smashing.github.io/
https://www.infoq.com/news/2017/04/avoid-alert-overload-services
https://blog.getambassador.io/the-importance-of-control-planes-with-service-meshes-and-front-proxies-665f90c80b3d

Summary

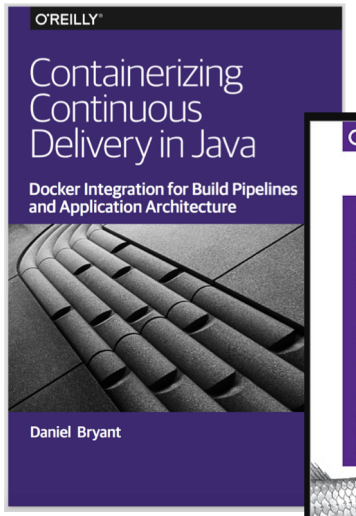
In summary

- Defining and effectively sharing the vision and goals is essential
  - Ask questions like the Cheshire Cat
- Using and agreeing on models reduces friction
  - Particularly around decision making and evaluation
- Optimising for feedback and learning is a core skill
  - Remember though, lots of what != why

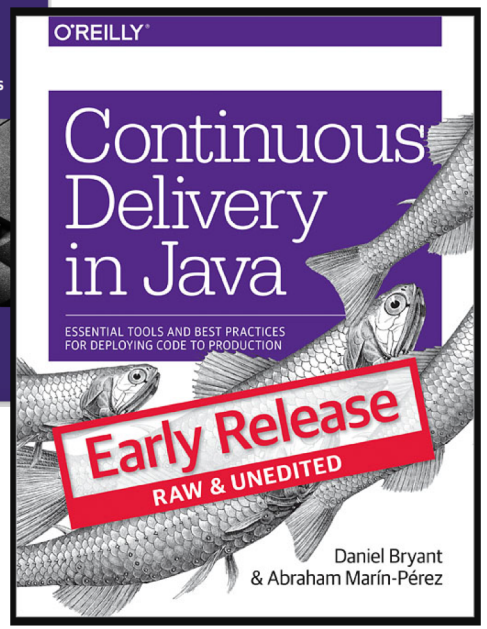
Thanks for listening…

Twitter: @danielbryantuk

Email: daniel.bryant@tai-dev.co.uk



bit.ly/2jWDSF7

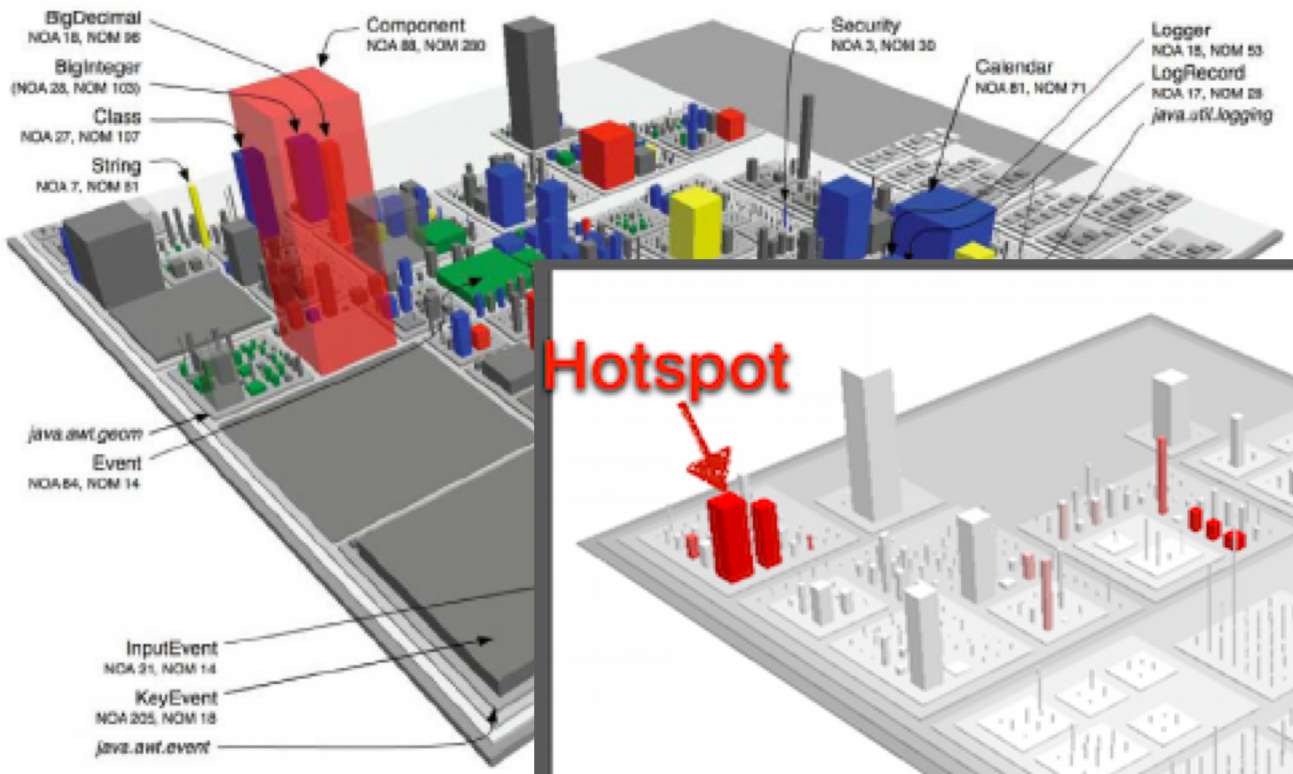


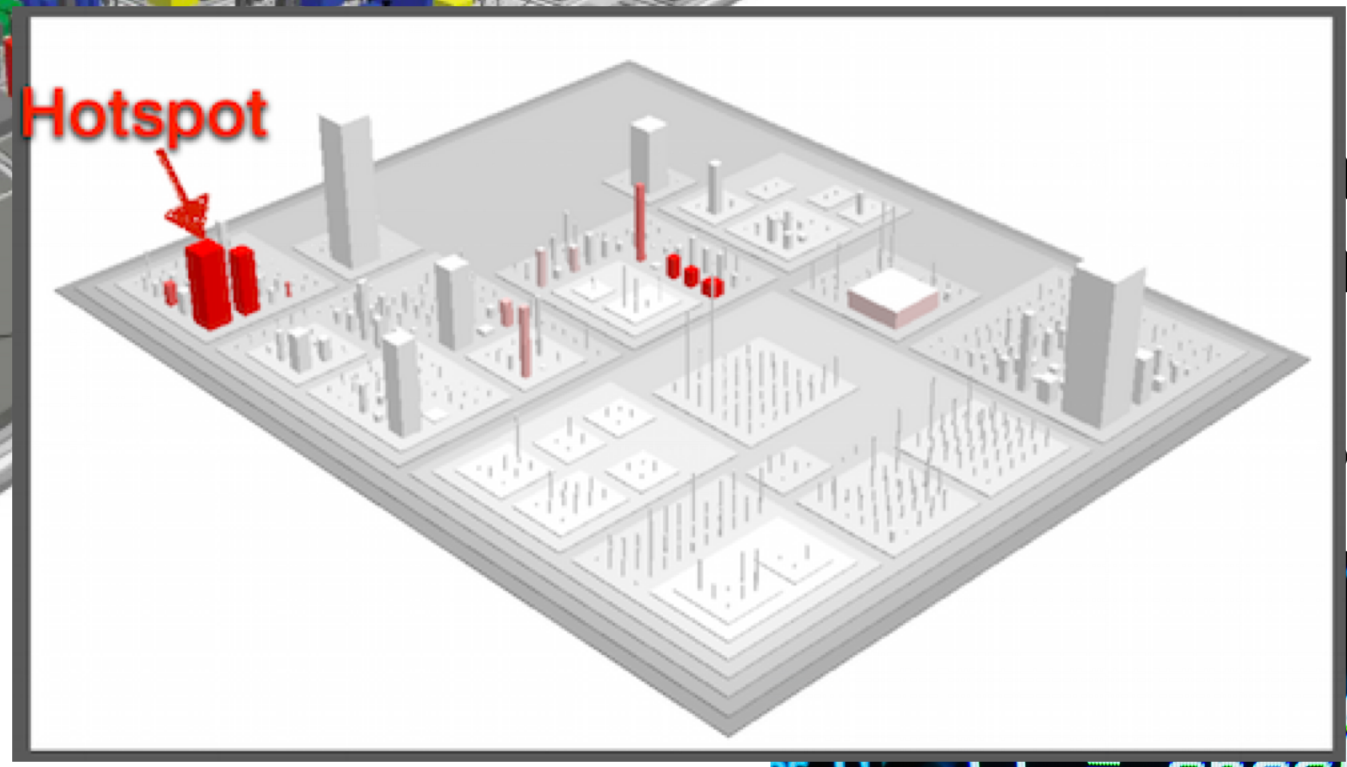
bit.ly/2BTjpir

Writing: https://www.infoq.com/profile/Daniel-Bryant

Talks: https://www.youtube.com/playlist?list=PLoVYf_0qOYNeBmrpjuBOOAqJnQb3QAEtM

Bonus



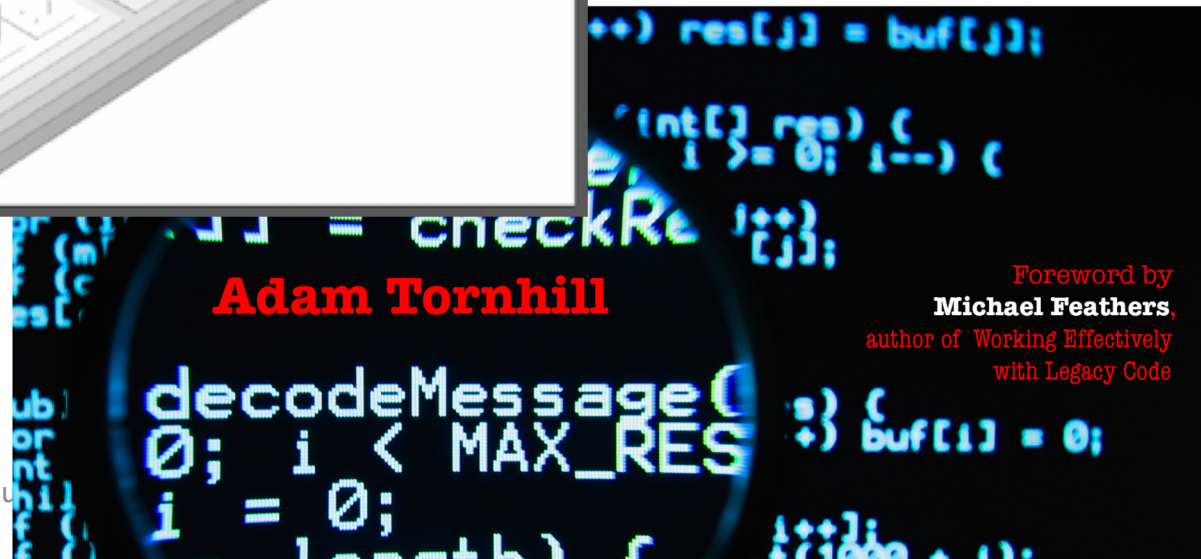


The Pragmatic Programmers

Investigator:
Date:
Case #:
Location:

...s a
...e

Use Forensic Techniques
Arrest Defects, Bottlenecks, and
Bad Design in Your Programs



https://www.slideshare.net/esug/codecity-esug2008
https://www.infoq.com/news/2015/03/code-as-a-crime-scene