THE APP BUSINESS

Future Role of the Architect

Reports of our extinction have been greatly exaggerated

Riccardo M Bennett-Lovsey

ITARC London, November 2016

A little about me

Senior Architect at "The App Business"

Began my career as a doctor of bioinformatics

Enterprise Fellow with the RSE

Student of the IDesign school of architecture

Co-organiser of the "Software Architect & Mentorship" Meetup

Contact via www.theappbusiness.com

Overview

The Role of the Architect

The Past

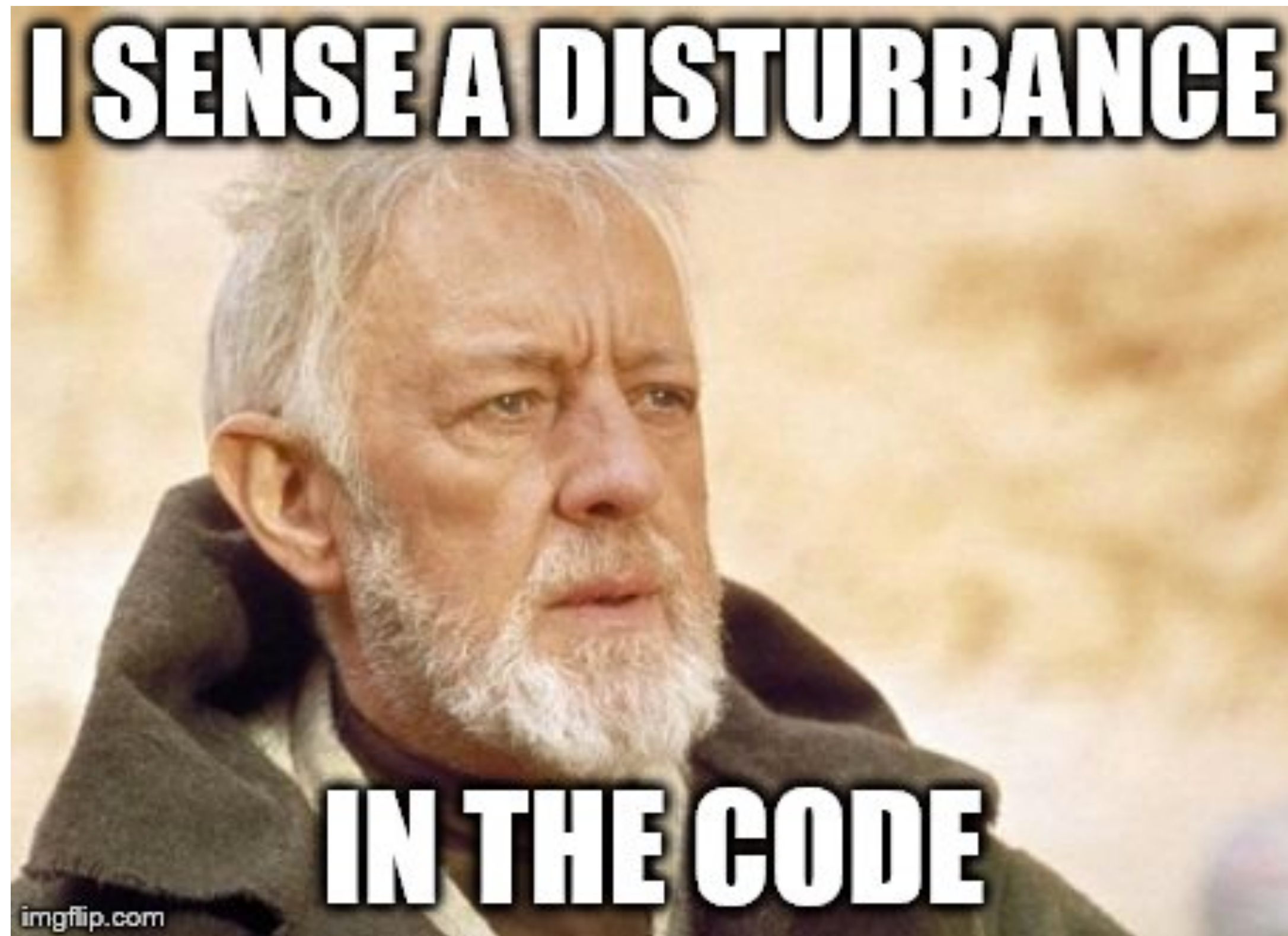## The Present

The Future

But first…
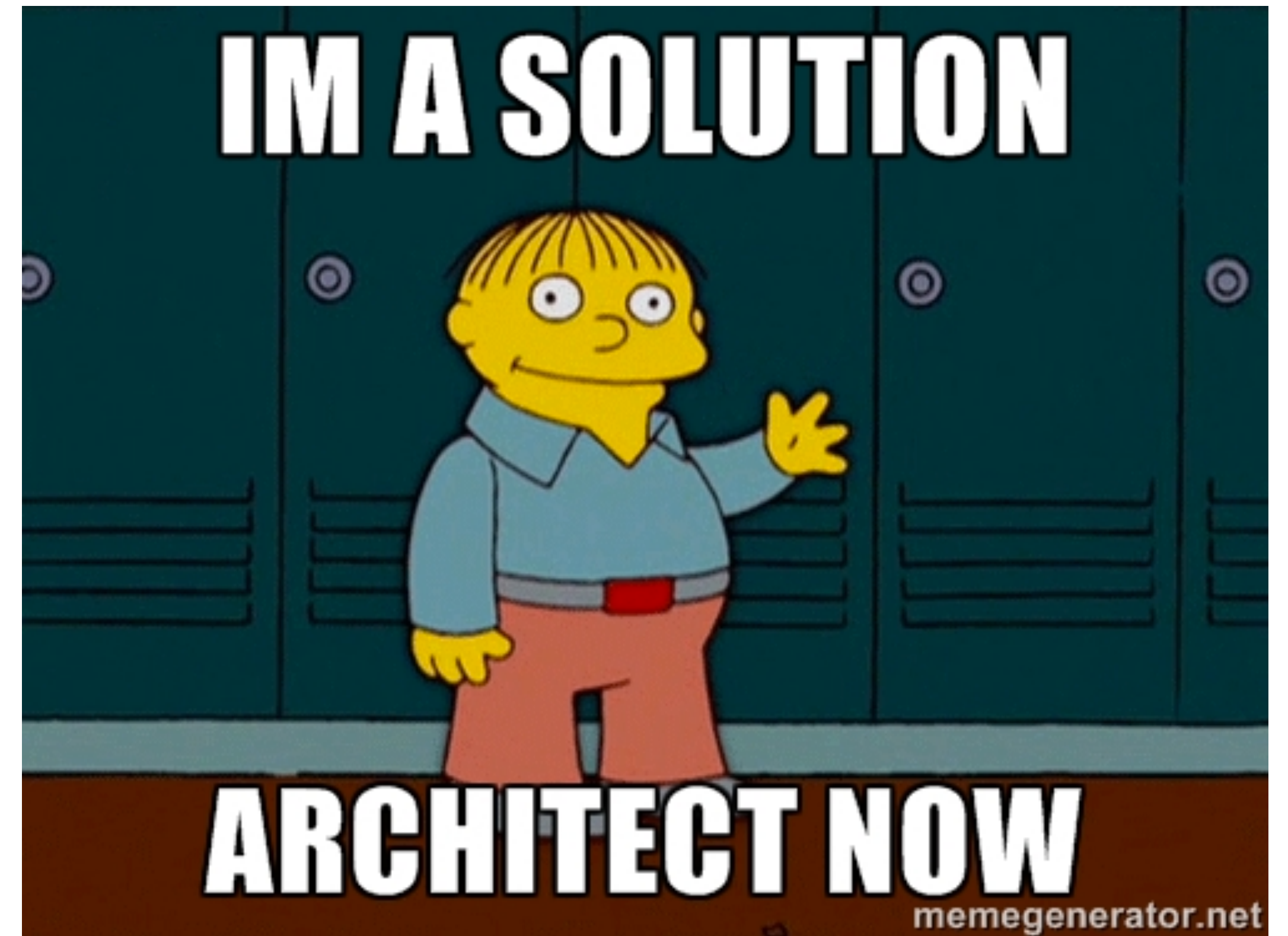
Warning:

Completely subjective

The Role of the Architect

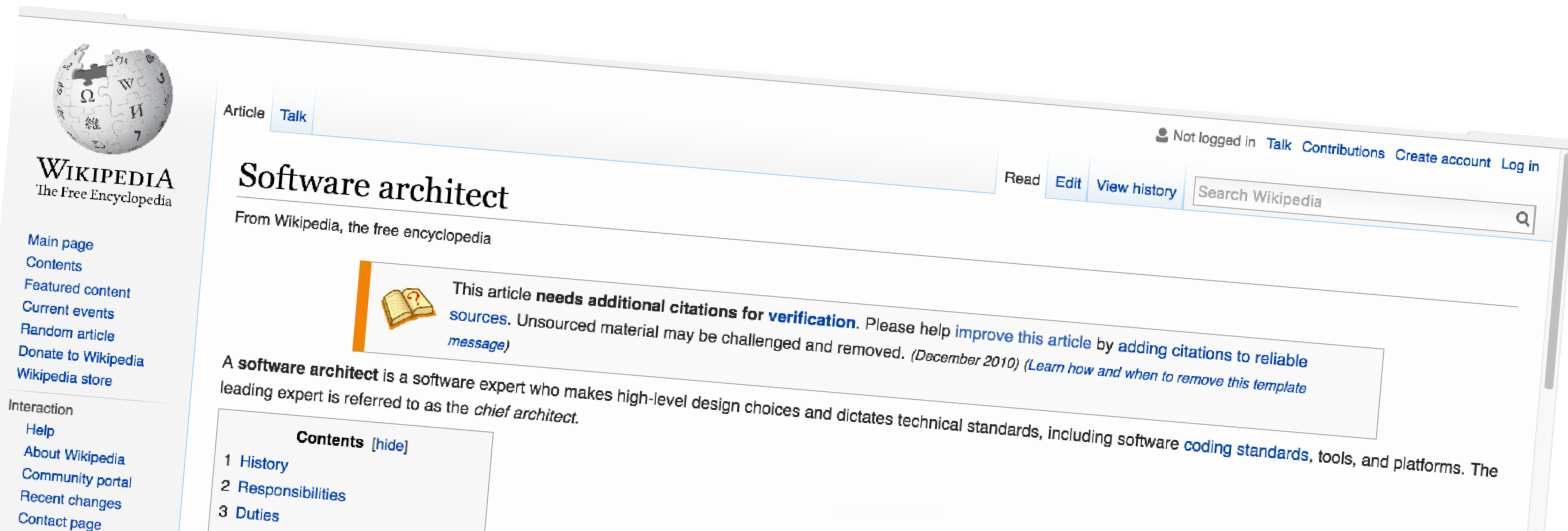# The Role of the Architect

How we like to see ourselves

How everyone else sees us

The Role of the Architect

A software architect is a software expert who makes high-level design choices and dictates technical standards, including software coding standards, tools, and platforms.

The Role of the Architect

Solution Architect

SOA Architect

Infrastructure Architect

Web Architect

Data Architect

Services Architect

Application Architect

Platform Architect

Technical Architect

Digital Architect

Domain Architect

System Architect

Cloud Architect

Software Architect

The Role of the Architect

"... can be summarised by any one of a collection of aphorisms, which (while equally nebulous) are still long enough to use in a job specification"

– Human Resources

The problem is…

We love to ruminate over what we <u>are</u>

Rarely think about what it is we actually <u>do</u>?

Compare with other Architects

Building Architect

Industrial Architect

Landscape Architect

Aerospace Architect

Aeronautical Architect

Nuclear Architect

Energy Architect

Naval Architect

Product Architect

Urban Architect

Unifying facets of an Architect

(my thoughts)

Identifying practical solutions to high-value problems

Integrating any number of parts* to build those solutions

Often the design, fabrication, configuration and validation of those parts

*define as necessary

Architectural Overheads

Constraints	Budget	Dependencies
Regulations	Communication	Politics
Governance	Diplomacy	Quality Control
Stakeholders	Requirements	Legacy Systems
Risk	Schedule	Process
Compliance	Technology	Planning

The Past

A quick history of software

Above the timeline (1940 → 2020):

- First digital computer built
- First mainframe systems
- "Decomposing Systems into Modules" published by Parnas
- World Wide Web invented
- .NET released
- Microsoft Azure released
- Azure Service Fabric released

Below the timeline:

- First software held in electronic memory
- First NATO Software Engineering Conference
- Software Engineering Institute established
- Java released
- Amazon EC2 released
- Docker released

Timeline markers: 1940 · 1950 · 1960 · 1970 · 1980 · 1990 · 2000 · 2010 · 2020

Where were the Architects?

For as long people have built software

there have been Architects

(whether they were called that or not)

Who were the Architects?

Chief Technician

System Engineer

Technical Lead

Lead Developer

Architect (occasionally)

Technical Manager

Development Lead

Core challenges have changed over time

Stopping valves burning out

Stopping the punchcards disintegrating

Access to the mainframe

Having enough processing power

Having enough memory

Having enough storage

Having enough bandwidth

Having the best platform

Having the best developers

Having the best testers

Having the best security

Service scalability

24/7 availability

Geo-redundancy

Software Crisis

Crisis? What Crisis?

(half a century of mediocrity)
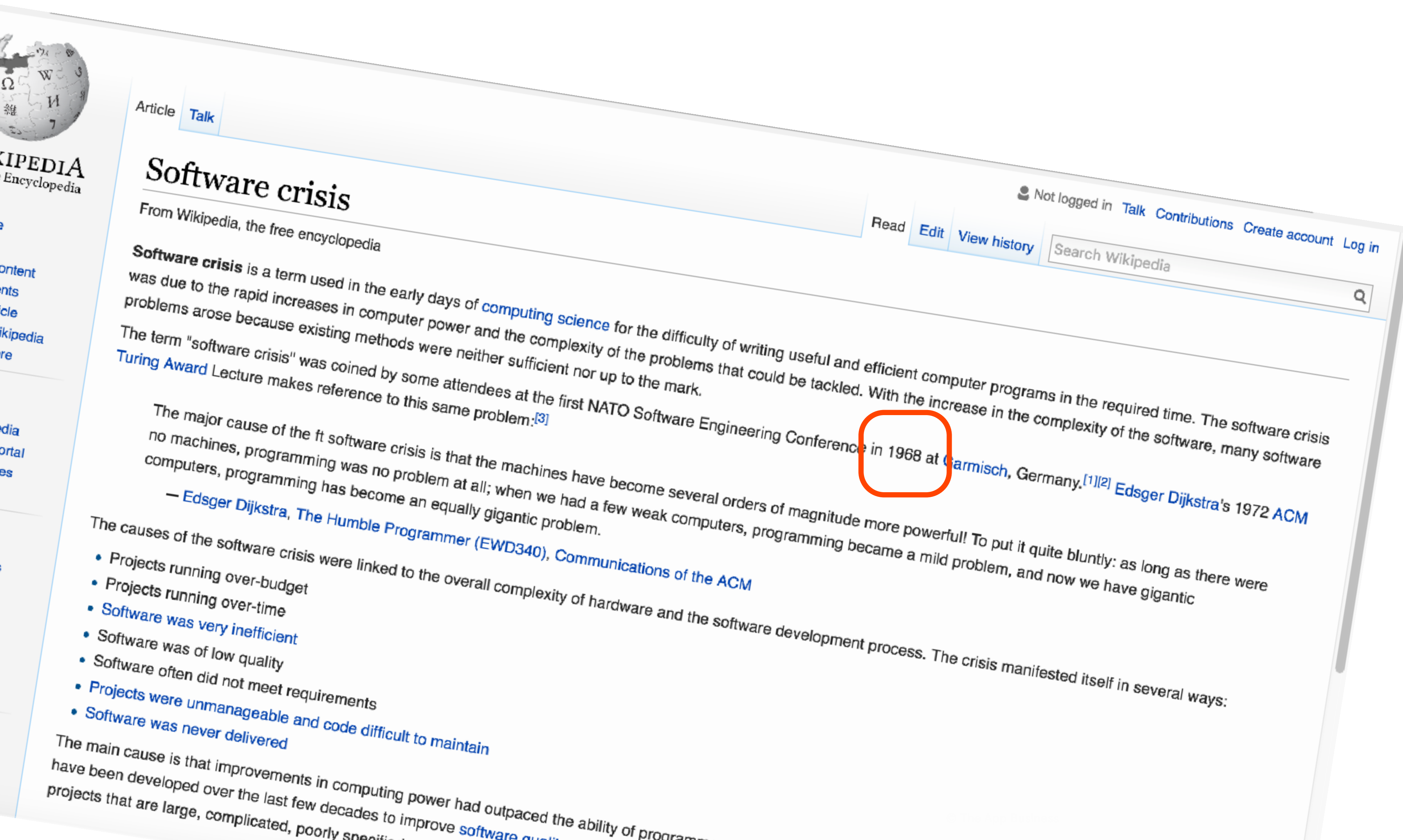
Projects over-budget

Projects running over-time

Inefficient software

Low quality deliverables

Unmet requirements

Unmaintainable products

Projects just not delivered

WIKIPEDIA
Encyclopedia

Article | Talk

Software crisis

From Wikipedia, the free encyclopedia

Read | Edit | View history | Search Wikipedia

Not logged in | Talk | Contributions | Create account | Log in

Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could be tackled. With the increase in the complexity of the software, many software problems arose because existing methods were neither sufficient nor up to the mark.

The term "software crisis" was coined by some attendees at the first NATO Software Engineering Conference in 1968 at Garmisch, Germany.[1][2] Edsger Dijkstra's 1972 ACM Turing Award Lecture makes reference to this same problem:[3]

The major cause of the ft software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

— Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

The main cause is that improvements in computing power had outpaced the ability of programmers have been developed over the last few decades to improve software quality, projects that are large, complicated, poorly specific

Crisis? What Crisis?

(half a century of mediocrity)

Projects over-budget

Projects running over-time

Inefficient software

Low quality deliverables

Unmet requirements

Unmaintainable products

Projects just not delivered

Article Talk

WIKIPEDIA
The Free Encyclopedia

Software crisis

From Wikipedia, the free encyclopedia

Not logged in Talk Contributions Create account Log in

Read Edit View history Search Wikipedia

Software crisis is a term used in the early days of computing science for the difficulty of writing was due to the rapid increases in computer power and the complexity of the problems that cou problems arose because existing methods were neither sufficient nor up to the mark.

The term "software crisis" was coined by some attendees at the first NATO Software Engineering Turing Award Lecture makes reference to this same problem:[3]

The major cause of the ft software crisis is that the machines have become several orders no machines, programming was no problem at all; when we had a few weak computers, pr computers, programming has become an equally gigantic problem.

— Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

The main cause is that improvements in computing power had outpaced the ability of programs have been developed over the last few decades to improve software quali projects that are large, complicated, poorly specifi

...e in 1968 at...

...s in the required time. The software crisis omplexity of the software, many software

many.[1][2] Edsger Dijkstra's 1972 ACM

quite bluntly: as long as there were and now we have gigantic

© The App Business

Crisis? What Crisis?

Why is there no "Electronics Crisis"?

… "Aeronautical Crisis"?

… "Construction Crisis"?

… "Manufacturing Crisis"?
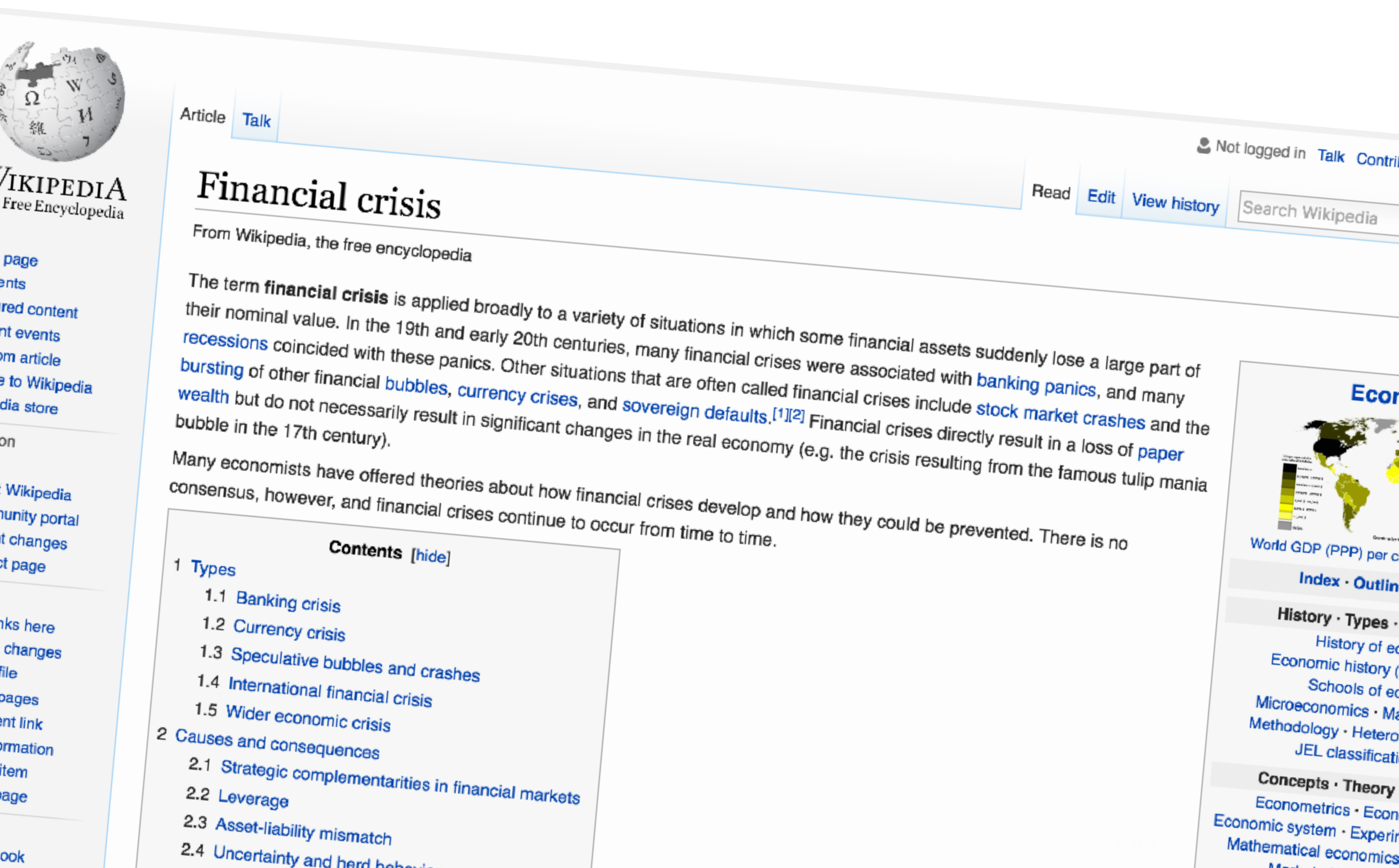
… "Mining Crisis"?

… "Transport Crisis"?

… "Agricultural Crisis"?

So, it's just us and...

Merchant Bankers

What went wrong?

What went wrong?

Did technology change too quickly?

Was software too unconstrained?

Did enough people know what they were doing?

Did enough people care?

Were enough people held accountable?

What went wrong?

Disciplines that do not hold their practitioners

to account are destined to stagnate or worse

The Present

Are we still in Crisis?

Projects over-budget

Projects running over-time

Inefficient software

Low quality deliverables

Unmet requirements

Unmaintainable products

Projects just not delivered

Why are we still in Crisis?

Software is still in adolescence

Is it art, science or engineering?*

No regulated licensing or industry standards

No formal qualification or apprenticeship required

*answer: engineering!

EULA: "Use at your own risk"

Would you accept the same warning on a bridge?

... or a plane?

... a car?

... a pace-maker?

... a toaster?

... a light switch?

... a doorbell?

Software has never had a "Revolution"

Scientific Revolution

First **Industrial** Revolution (Machines & Manufacturing)

Medical Revolution

Digital Revolution

1500   1600   1700   1800   1900   2000

First **Agricultural** Revolution

Second **Industrial** Revolution (Technology & Transport)

Second **Agricultural** Revolution (Green Revolution)
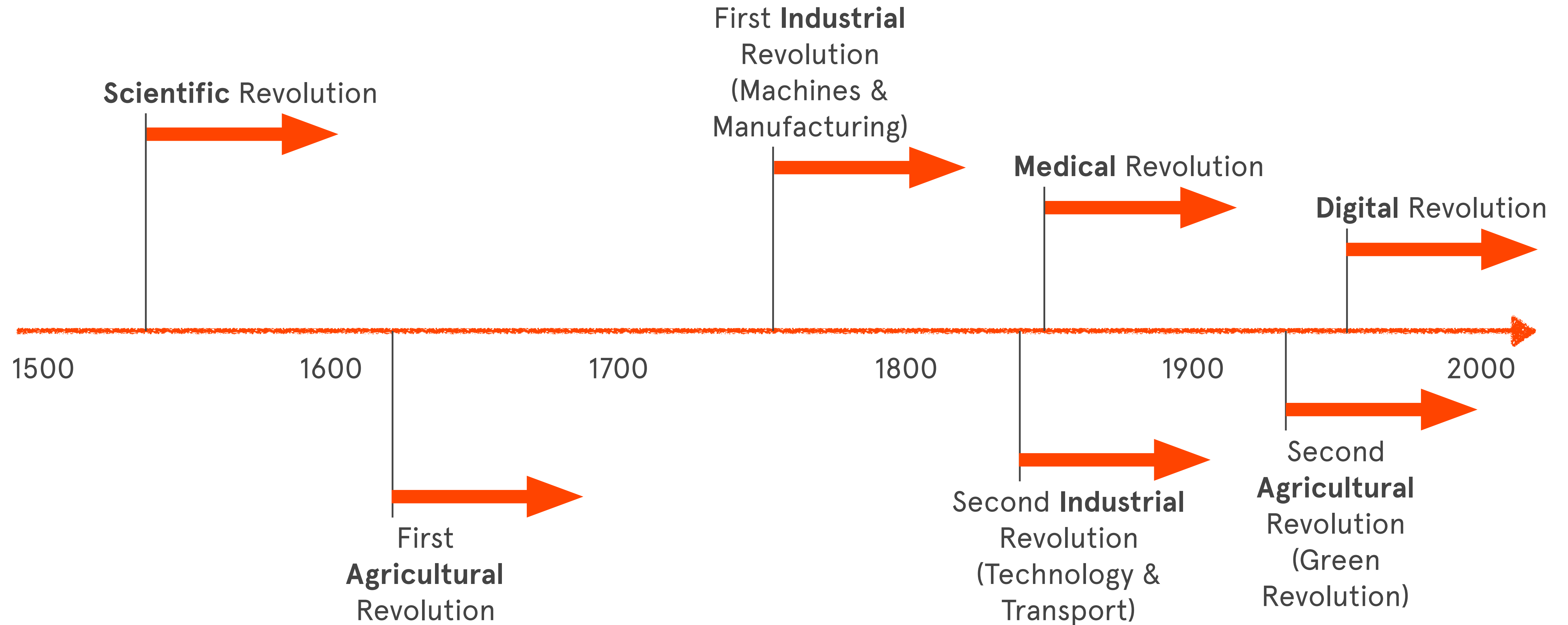
Yet, we are all software companies now

THE WALL STREET JOURNAL.

Home   World   U.S.   Politics   Economy   Business   Tech   Markets   Opinion   Arts   Life

How Many Thanksgiving Meals Can You Stand?

OFF DUTY TRAVEL
Breakfast Tacos: The 5 Best Places to Find Them in ...

MIND & MATTER
Empathy by the Book: How Fiction Affects Behavior

ESSAY

Why Software Is Eating The World

By MARC ANDREESSEN

August 20, 2011

This week, Hewlett-Packard (where I am on the board) announced that it is exploring jettisoning its struggling PC business in favor of investing more heavily in software, where it sees better potential for growth. Meanwhile, Google plans to buy up the

Who should hold themselves accountable?

Answer: we should

(because no one else will)

We must drive the change

We must drive the change

Set an example to the newbies

Become the standard-bearers

Take responsibility for our industry

We must drive the change

... otherwise, why else are we here?

The Future

Remaining relevant

"Revolution" begets industrialisation

Industrialisation begets automation

Automation begets redundancy

What else must we do?

What we always do:

Adapt or perish

Recall how challenges have changed over time

Stopping valves burning out

Stopping the punchcards disintegrating

Access to the mainframe

Having enough processing power

Having enough memory

Having enough storage

Having enough bandwidth

Having the best platform

Having the best developers

Having the best testers

Having the best security

Service scalability

24/7 availability

Geo-redundancy

Recall how challenges have changed over time

Eventually technology ceases to be the limitation

Wardley Value Maps

© The App Business

Wardley Value Maps

© The App Business

Copyright © Simon Wardley
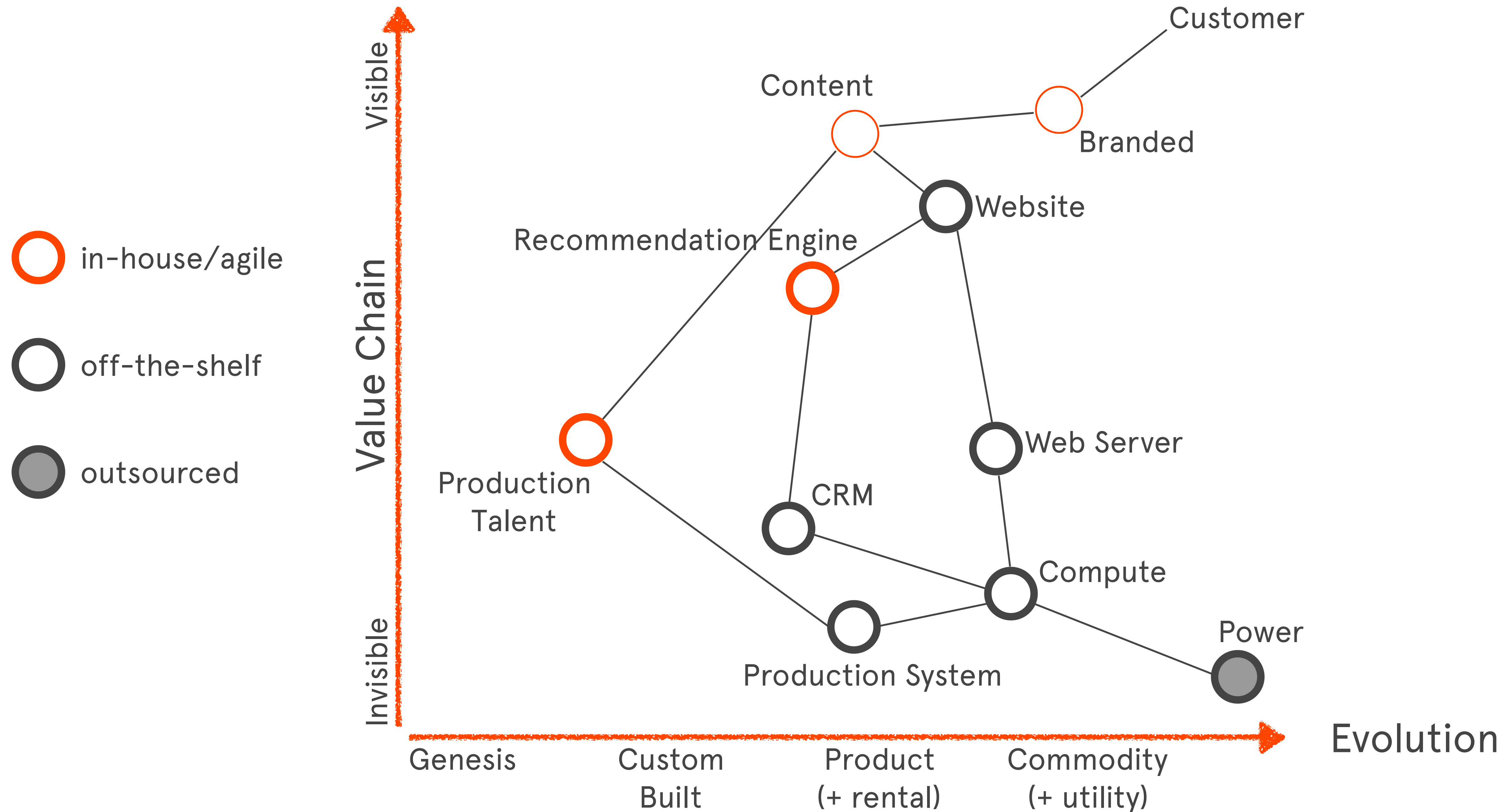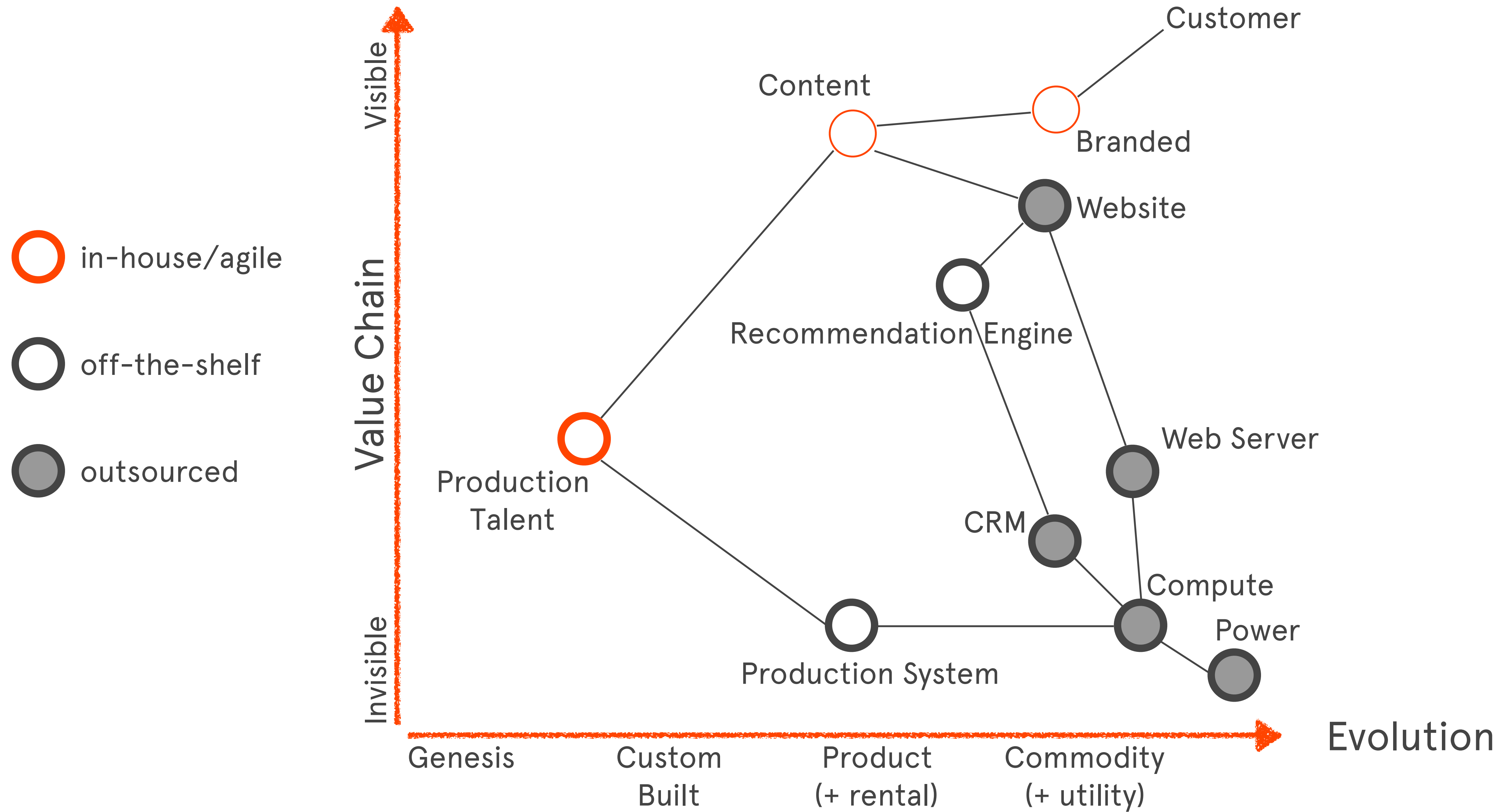
Already commoditised

Communication	Scale	Support
Storage	Resilience	Logging
Compute	Security	Analytics
Search	CRM	Office
Maps	ERP	Payment
Availability	Business Intelligence	Vending

Already commoditised

Even coding is becoming commoditised

Exhibit A: Azure Logic Apps

Exhibit B: Azure Service Fabric
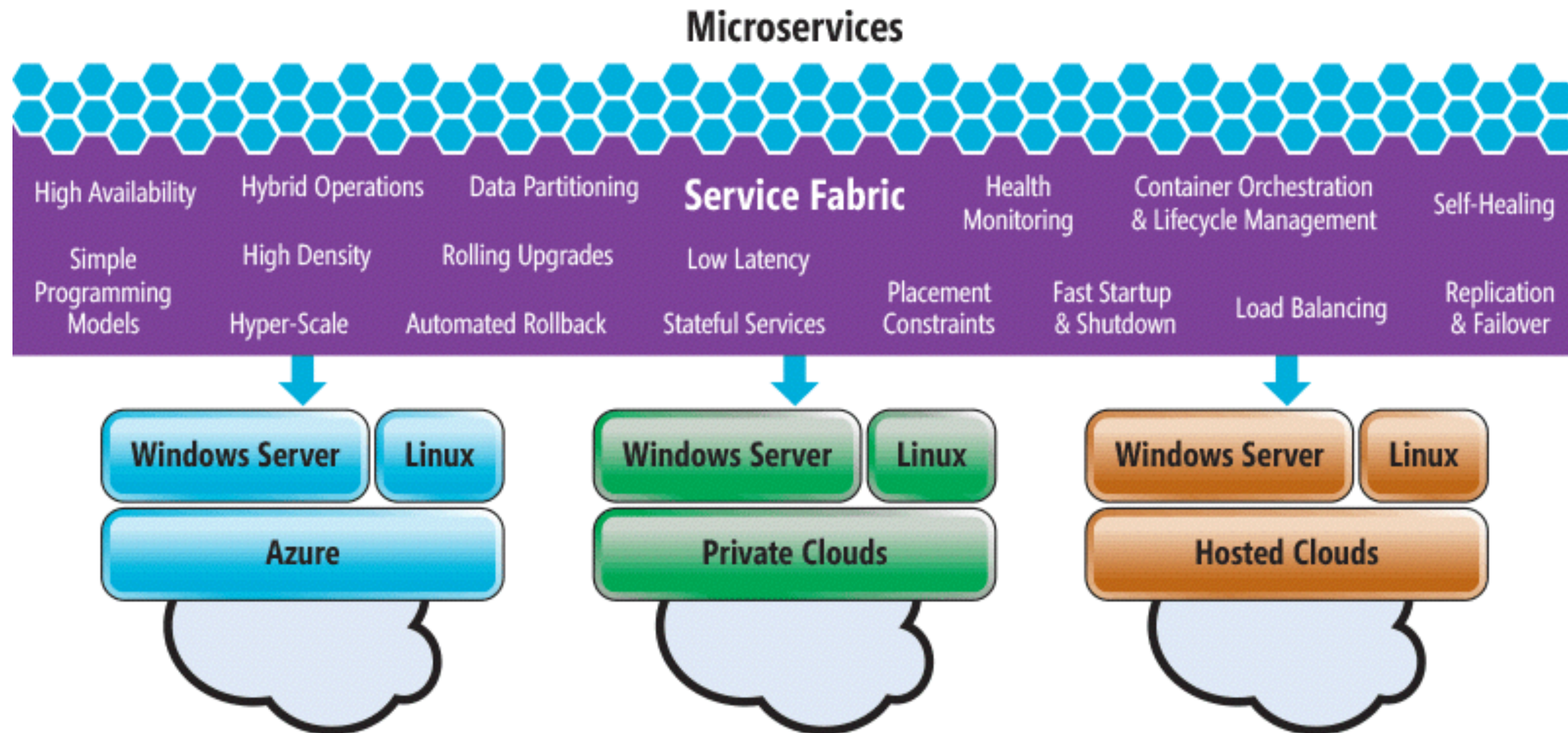
What does all this mean for us?

Recall the unifying facets of an Architect

Identifying practical solutions to high-value problems

Integrating any number of parts* to build those solutions

Often the design, fabrication, configuration and validation of those parts

*define as necessary

We must redefine the "parts" that we integrate

Scripts

Functions

Libraries

Frameworks

Servers

Systems

Products

Services

Utilities

Capabilities

We must redefine the "parts" that we integrate

Things will always need integrating

Look ahead and drive innovation

Move beyond just technology

To realms of business and enterprise

Thinking in terms of "capabilities"

Opportunities are coming faster than ever

Never been a better time

The world needs Architects more than ever

© The App Business

Thank You