Beginning with Microservice

Wayne Yeh @ 2019-06-28

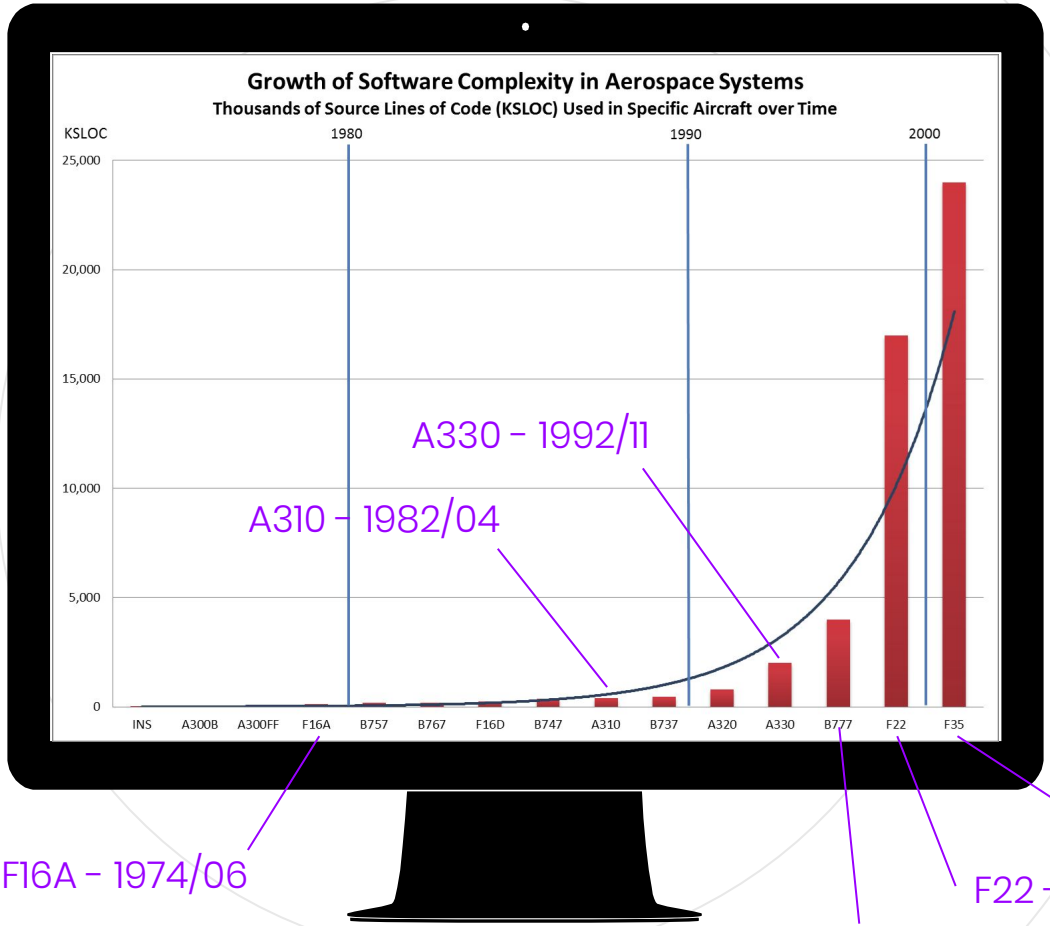
Outline

- History
- Microservice
- Why microservice?

1

History

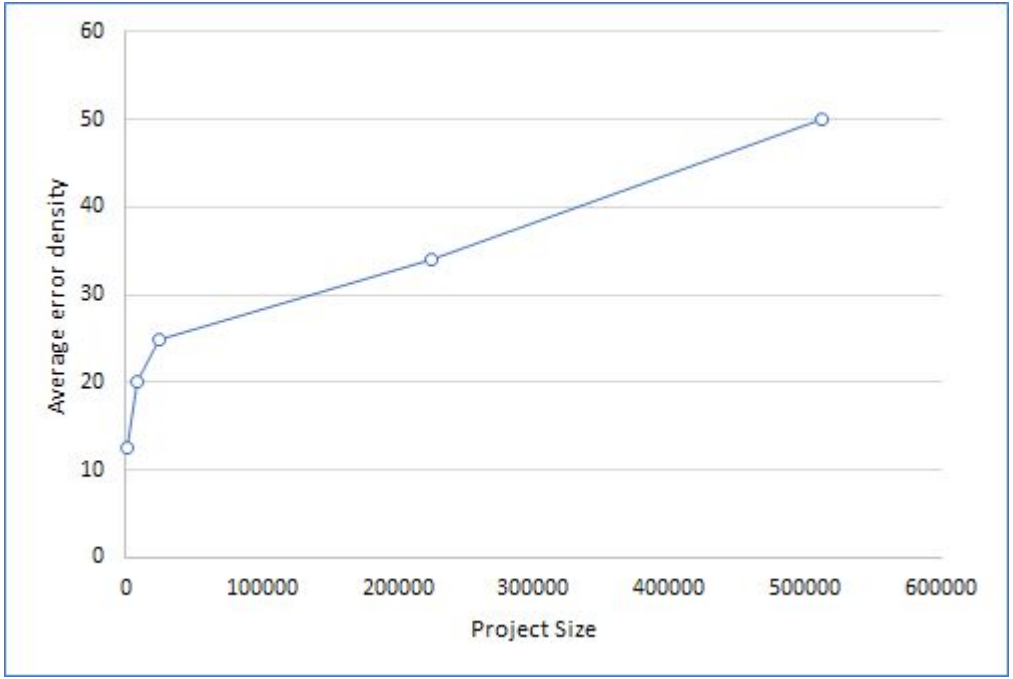
Growth of Software Complexity



https://savi.avsi.aero/about-savi/savi-motivation/

Pain when project growth

Project size (lines of code)	Average error density (per 1000 lines of code)
Less than 2K	0 - 25 errors
2K - 16K	0 - 40 errors
16K - 64K	0.5 - 50 errors
64K - 512K	2 - 70 errors
512K and more	4 - 100 errors



https://hownot2code.com/2017/11/08/search-for-bugs-in-code-at-the-early-stage/

Big things cause troubles

People are trying to solve these mess. Make things small could be good, but what's the price we need to pay?

History of “Microservice”

2005

"Micro-Web-Services" is introduced by Dr. Peter Rodgers.

2011

Term “microservice” is used in a software architecture.

2012

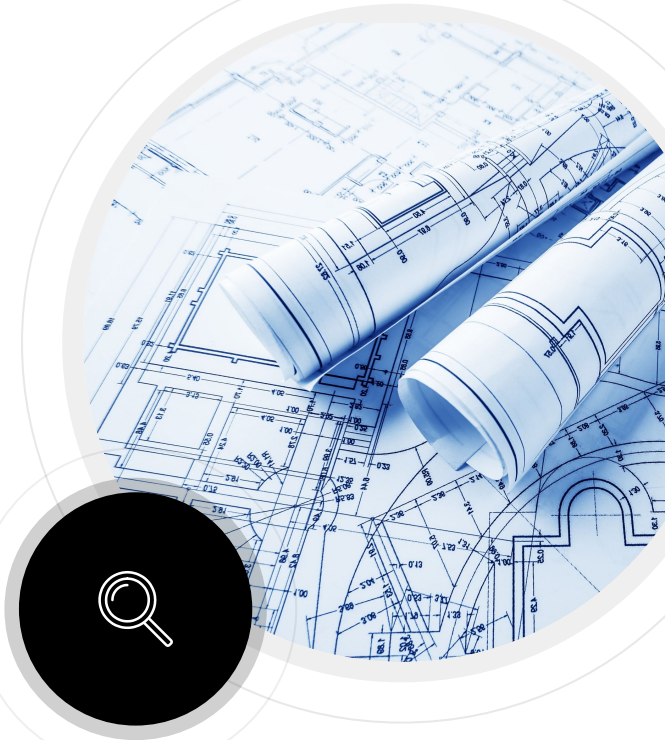
James Lewis presented similar ideas as a case study.

2012

"microservices" is decided to be the most appropriate name.

2014

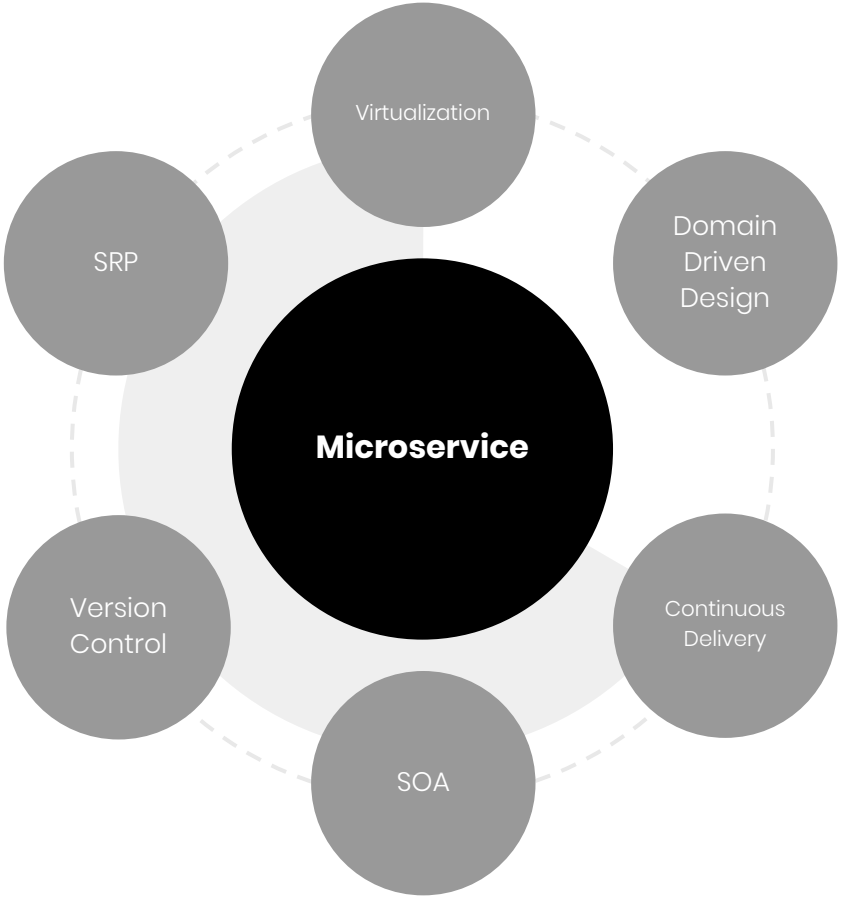
“Microservice” is proposed by Martin Fowler and James Lewis.



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

https://martinfowler.com/articles/microservices.html

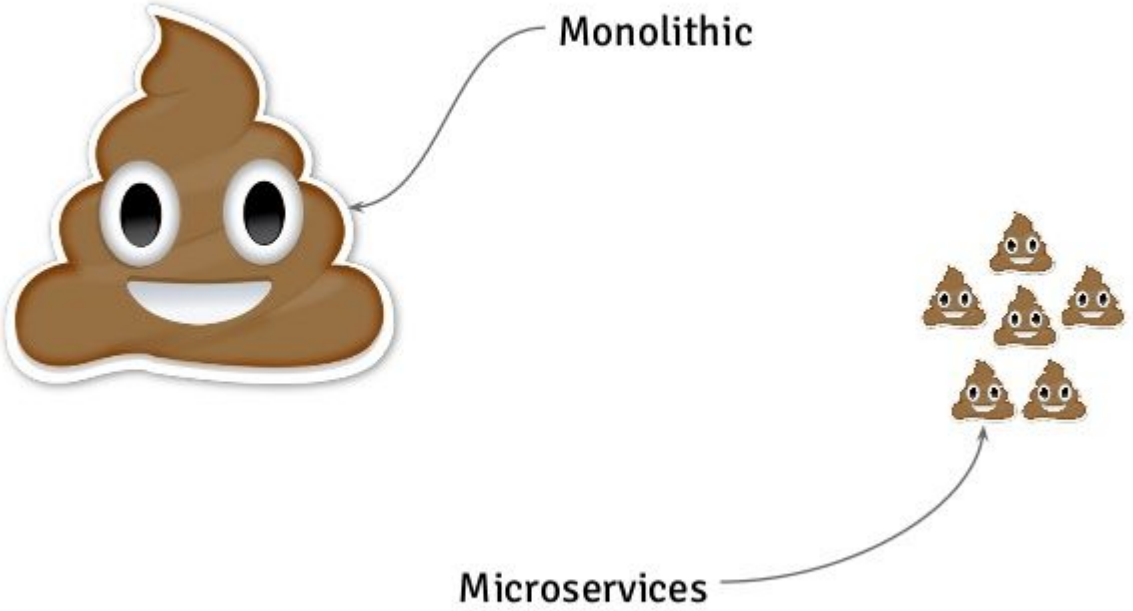
Microservice is not a new idea



2

Microservice

Monolithic vs Microservices



@alvaro_sanchez

odobo

https://lostechies.com/andrewsiemer/2016/01/11/testing-microservices/

What is "**small**"?

> ***Microservices are small, autonomous services that work together.***
>
> ***微服務是協同運作的小型自主服務。***

No Silver Bullet

Microservice is not a silver bullet for solving all problems. It solve problems while creating new problems.

There is always a price to pay.

3

Why microservice?

What's the pros. and cons. of microservice?

https://learning.oreilly.com/library/view/building-microservices/9781491950340/

Key Benefit:
Technology Heterogeneity
技術異質性

Technical Stack

Choose the most appropriate technical stack for every service.

Acceptable Risk

Restricting the impact and risk while adopting a new technique.

Quick Adaptation

Adapt the new technique more quickly as its impact is limited.

Entry Barrier

Embracing multiple techniques brings corresponding overheads.

Shared Libraries

Developing utilities could ease the services with the same tech., but block others.

It's all about finding the right balance.

Key Benefit: Resilience 彈性

Bulkhead

Failure can be limited in a specific area when it is a fine grained microservice.

Everything could be failed

Be aware that machine could be failed as well as network.

Key Benefit: Scaling 擴展

Scale Separately

Scale things that need to be scaled only.

Cost Control

Put services and components in its best size.

On-demand

No need to provision instances that currently useless.

Key Benefit:
Ease of Deployment 容易部署

Huge deployment is hard

One line of change makes the whole application to be re-deployed, and brings complexity and fear in the same time.

Late deployment increase risk

Due to deployment is complex, people intend to reduce the times of deployment, which makes problem worse.

Clean and fast

On the contrast, microservice deployment can be relatively simple, easy, and resilient. Rolling back could be easy, too.

Key Benefit: Organizational Alignment
組織調校

Small team works better

Big team with many teammates and large codebase could cause troubles. Small teams and small codebase often have a better productivity.

Conway's Law

Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.

> If you have four groups working on a compiler, you'll get a 4-pass compiler.
>
> Eric S. Raymond

Key Benefit: Composability 組合性

Reuse by services

When it comes to monolithic application, trying to re-use functions in the application can be complex. However, it is the nature of microservice to re-use codes and functionalities.

Device diversity in present time

We need a more resilient architecture to fulfill the present situation that clients come from much more medias than ever.

Key Benefit:
Optimizing for Replaceability
最佳化可替換性

Monolithic is hard to replace

Can you imagine to replace an important legacy system which contains millions of lines of codes?

Replaceability

Microservice has small amount of codes which should be easy to re-write or replace.

Technologies change overtime

Teams using microservice approaches are comfortable with completely rewriting services when required, and just killing a service when it is no longer needed.

No Silver Bullet

Microservice sounds good, but it comes with more challenges such as deploying, monitoring, and testing. Don't be too surprised when you encounter issues like distributed transaction or CAP.

There is always a price to pay.

https://microservices.io/patterns/index.html

Thanks!