# Tackling the ever changing *essential* complexities of engineering software

Gail C. Murphy
University of British Columbia



@gail_murphy
gail_murphy@social.sigsoft.org

Images not available for reuse

# Bridges

Holzbrücke Rappersweil-Hurden (wooden timber)

1500 BCE

Arkadiko

1300 BCE

Zhaozhou

600 CE

Wetherby

1200 CE

Akashi Kaikyo

1998 CE

Images not available for reuse

## Bridges

Holzbrücke Rappersweil-Hurden (wooden timber) — 1500 BCE

Arkadiko — 1300 BCE

Zhaozhou — 600 CE

Wetherby — 1200 CE

Akashi Kaikyo — 1998 CE

## Software

1948 CE — Manchester Baby

1957 CE — Fortran compiler

~1970 CE — Unix

1998 CE — Mosaic graphical web browser (also in 1994 Shor's algorithm for quantum computers)

2007 CE — iPhone and iOS

Images not available for reuse

## Bridges

Holzbrücke Rappersweil-Hurden (wooden timber)
1500 BCE

Arkadiko
1300 BCE

Zhaozhou
600 CE

Wetherby
1200 CE

Akashi Kaikyo
1998 CE

~3500 years

## Software

1948 CE
Manchester Baby

1957 CE
Fortran compiler

~1970 CE
Unix

1998 CE
Mosaic graphical web browser (also in 1994 Shor's algorithm for quantum computers)

2007 CE
iPhone and iOS

Images not available for reuse

## Bridges

- 1500 BCE — Holzbrücke Rappersweil-Hurden (wooden timber)
- 1300 BCE — Arkadiko

- 600 CE — Zhaozhou
- 1200 CE — Wetherby
- 1998 CE — Akashi Kaikyo

~3500 years

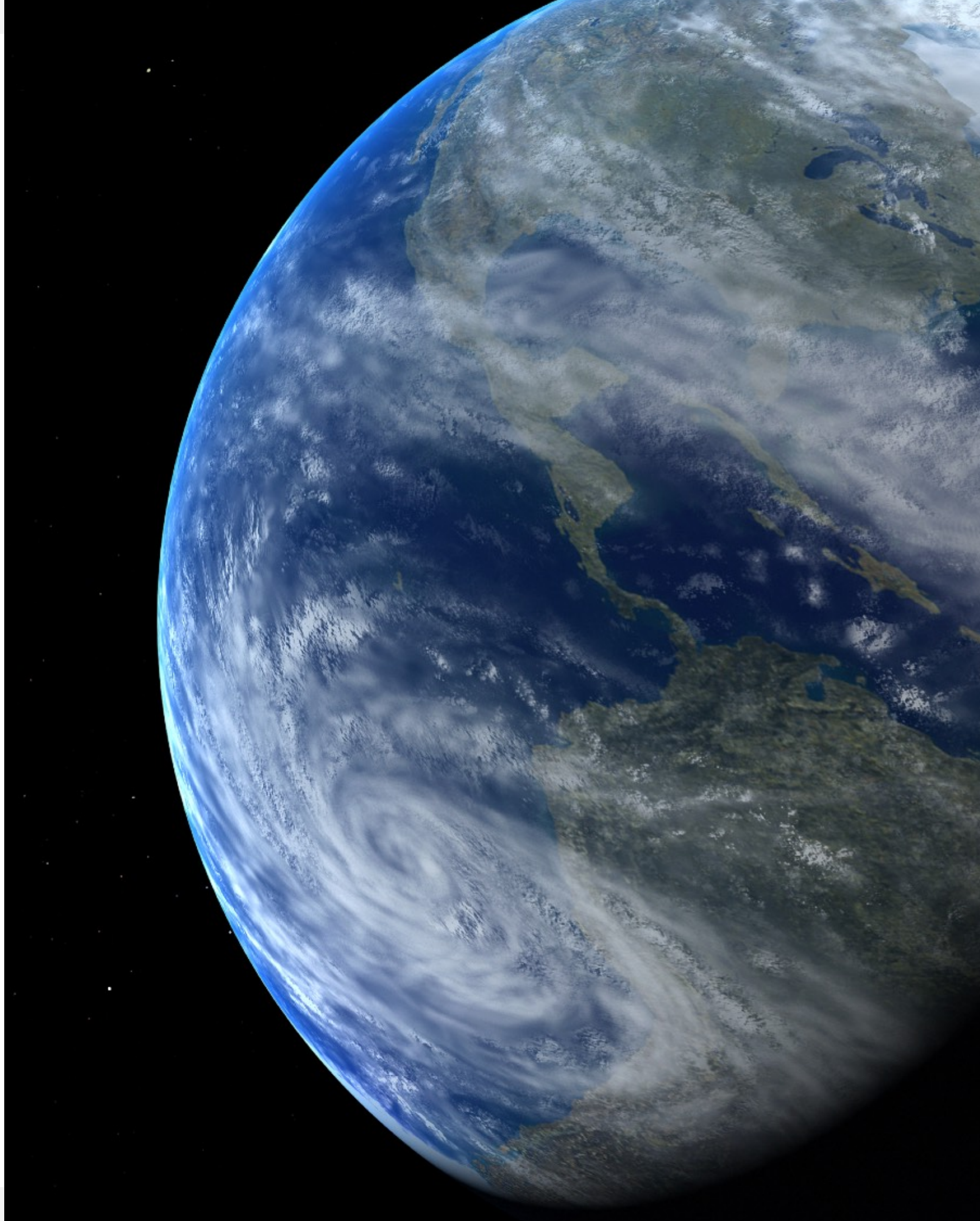
## Software

- 1948 CE — Manchester Baby
- 1957 CE — Fortran compiler
- ~1970 CE — Unix
- 1998 CE — Mosaic graphical web browser (also in 1994 Shor’s algorithm for quantum computers)
- 2007 CE — iPhone and iOS

~60 years

64% of the global population is connected to the internet

Software runs infrastructure,
disrupts industries,
is changing the nature of work, and helping to improve the quality of life



Images not available for reuse

Software engineering involves…

"multi-person multi-version development"

—Brian Randell

Over the last 50+ years, has software engineering research focused enough on what are…

THE ESSENTIAL COMPLEXITIES* OF DEVELOPING SOFTWARE?

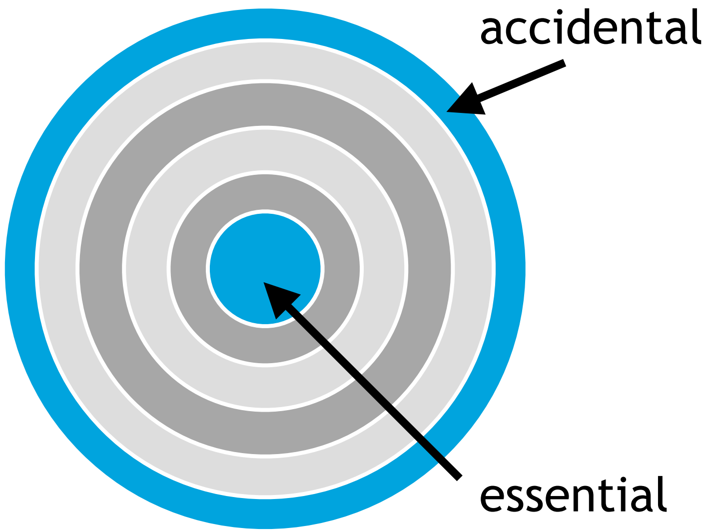
* per Fred Brooks

Too much of our focus is on the building blocks (the “accidental”) of software instead of the whole

]

Images not available for reuse

# Take-aways

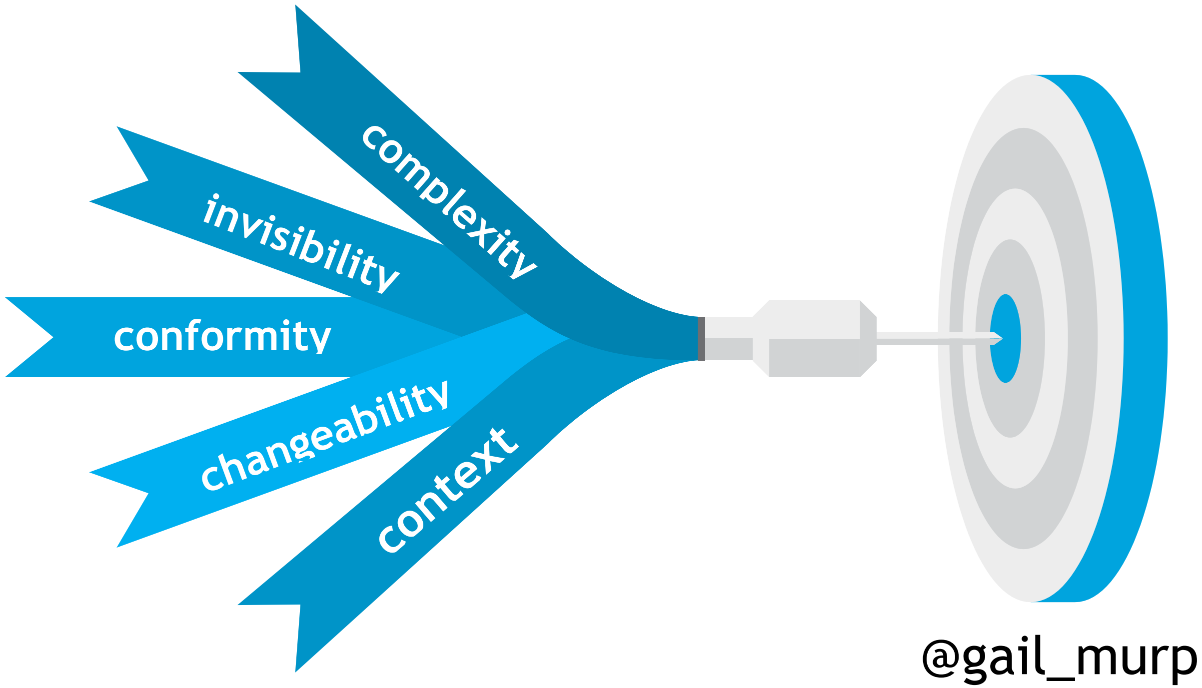



Move from foci on *accidental* complexities to more study about the *essential* complexities of growing software

Consider more…

holistic, longitudinal and interdisciplinary study of software in-situ and at scale…

which has implications for funding and research assessments

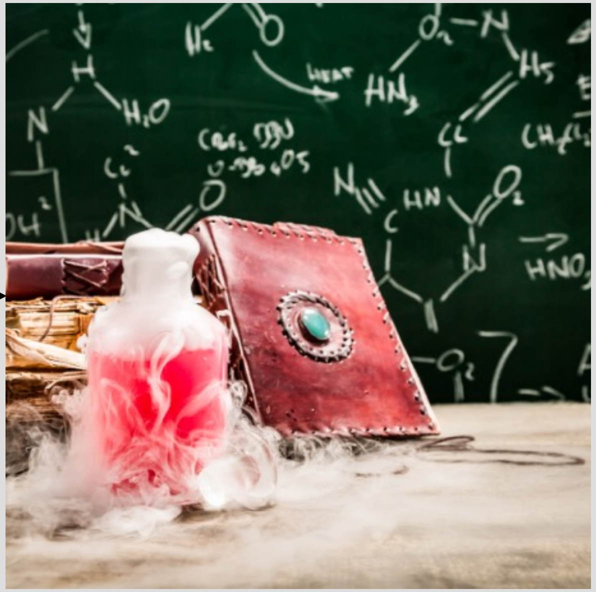



@gail_murphy
gail_murphy@social.sigsoft.org

# “No silver bullet” recap

Images not available for reuse

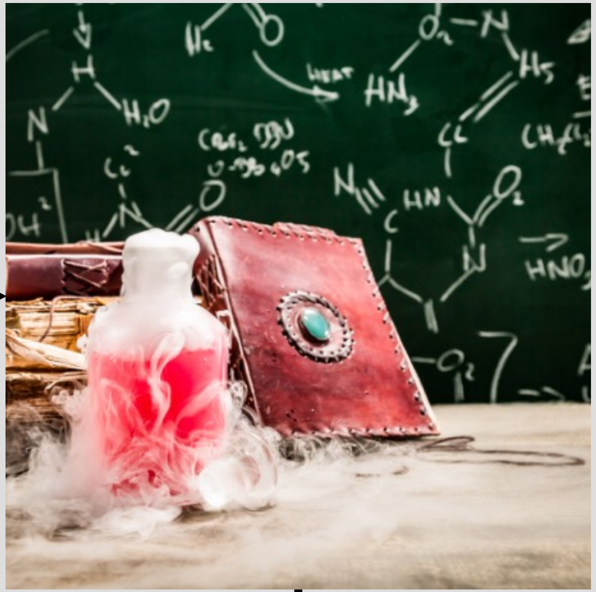
“No silver bullet” recap



The last 25 years

Images not available for reuse

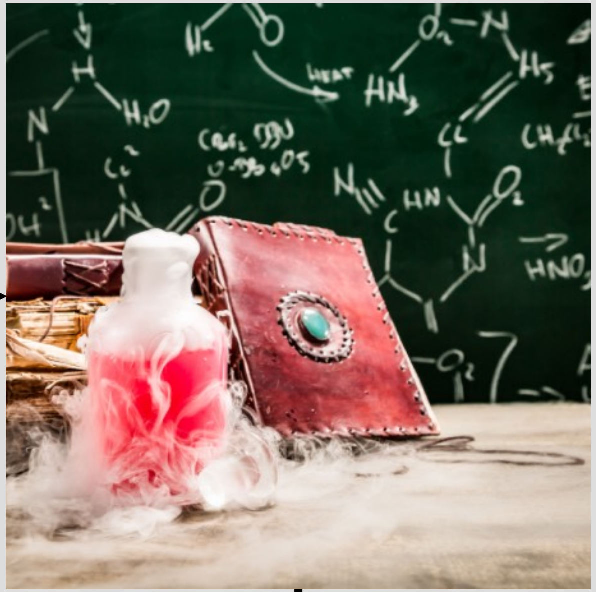
"No silver bullet" recap



The last 25 years

Essential complexities in 2023

“No silver bullet” recap



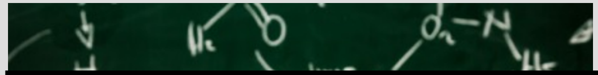
The last 25 years

Research opportunities


Essential complexities in 2023

Images not available for reuse

“No silver bullet” recap



*Disclaimer*

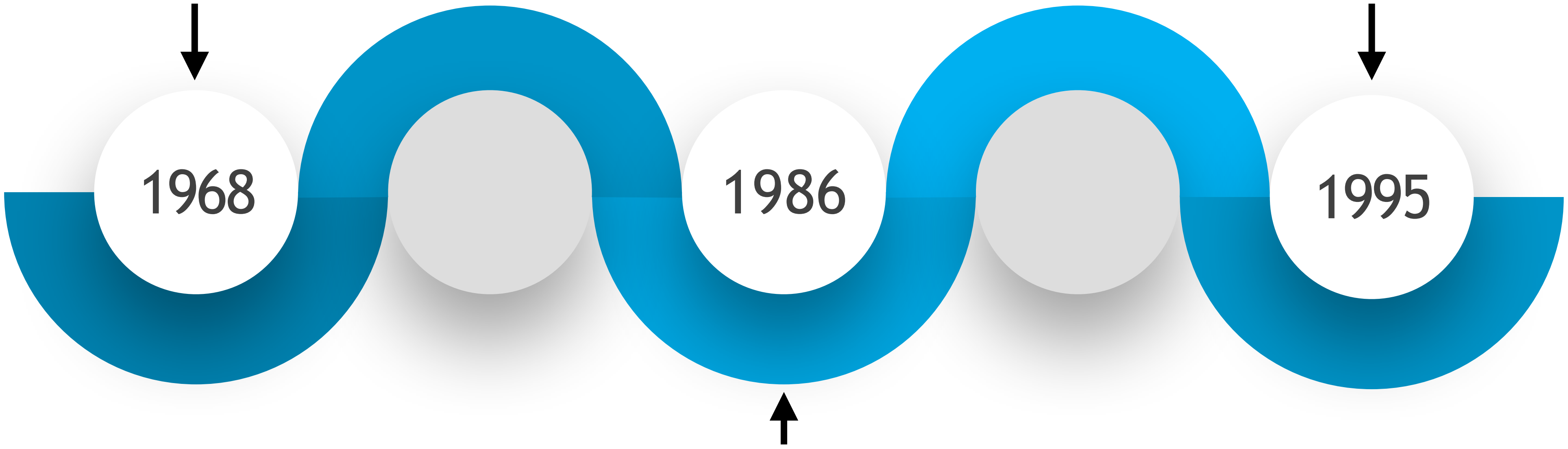
*I will do my best to accurately reflect the work of others, especially, Frederick P. Brooks Jr., but any inaccuracies are due to my own interpretations*

*I will raise more questions than I answer*

Research opportunities

NATO Software Engineering Conference

“No Silver Bullet” Refined (10 years later)

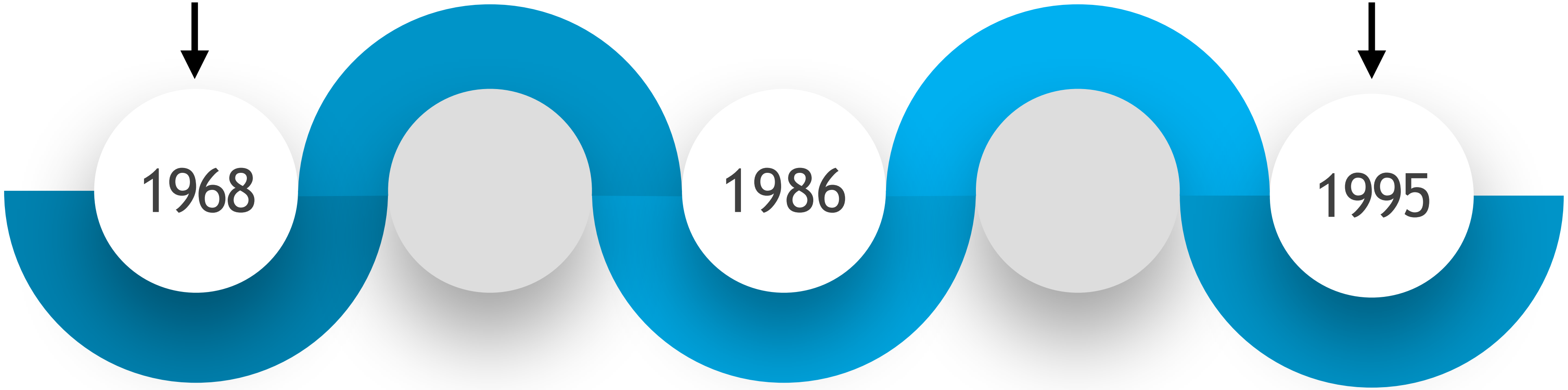



“No Silver Bullet — Essence and Accident in Software Engineering
Invited Paper
by Frederick P. Brooks Jr.

“But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity.”

NATO Software Engineering Conference

"No Silver Bullet" Refined (10 years later)




"No Silver Bullet — Essence and Accident in Software Engineering
Invited Paper
by Frederick P. Brooks Jr.

"But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity."

NATO Software Engineering Conference

1968

“No Silver Bullet” Refined (10 years later)

1995

“No Silver Bullet — E

Software Engineering


by Frederick P. Brooks Jr.

“But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity.”

Images not available for reuse

# Brook's Essential Complexities (1986)

## Complexity

No two parts are alike
Many parts needed

## Conformity

Software most conformable
Complexity from conforming

Images not available for reuse

# Brook’s Essential Complexities (1986)

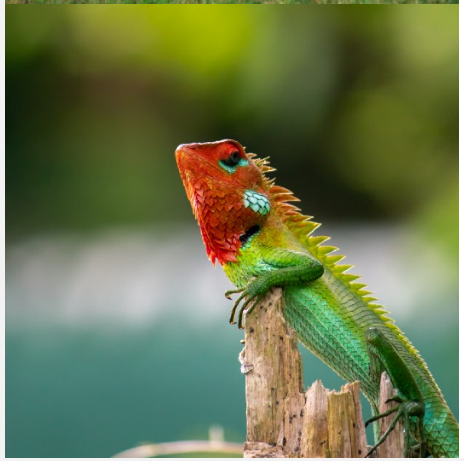
## Changeability

Software is constantly subject to change and is infinitely changeable

## Invisibility

“Software is invisible and unvisualizable”

# Brook’s Essential Complexities Remain (1995)

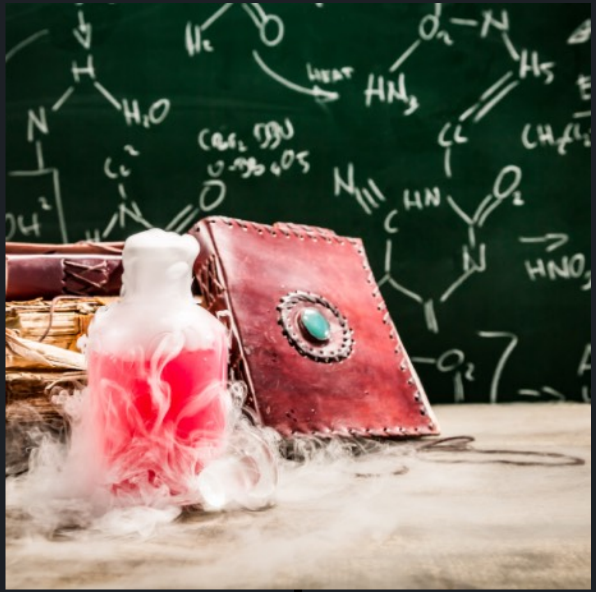


Improvements have addressed accidental (incidental) complexities

- Reuse
- Object-Orientation
- AI
- Verification
- Environments and Tools

Images not available for reuse

"No silver bullet" recap



The last 25 years

Research opportunities


Essential complexities in 2023

Images not available for reuse

# Some notable advances in the last 25 years

01 Open source

02 Cloud

03 Automation

04 Generation

# 01 Open Source

Software Supply Chain 2022*

| Ecosystem | Total Projects | Annual Request Volume | Avg. Versions per Project |
|---|---|---|---|
| Java (Maven) | 492k | 675B | 19 |
| JavaScript (npm) | 2.06M | 2.1T | 14 |

Open source enables significant reuse, easing initial development

But, use of open source is not zero cost…

Java application ~ 148 dependencies
Java project - 10 updates per year

… means application developers are tracking ~1500 dependency changes per year per project

* From sonatype, 8th annual State of the Software Supply Chain

# 01 Open Source

Software Supply Chain 2022*

| Ecosystem | Total Projects | Annual Request Volume | Avg. Versions per Project |
|---|---|---|---|
| Java (Maven) | 492k | 675B | 19 |
| JavaScript (npm) | 2.06M | 2.1T | 14 |

Open source enables significant reuse, easing initial development

But, use of open source is not zero cost…

Java application ~ 148 dependencies
Java project - 10 updates per year

… means application developers are tracking ~1500 dependency changes per year per project

* From sonatype, 8th annual State of the Software Supply Chain

# 01 Open Source

Software Supply Chain 2022*

| Ecosystem | Total Projects | Annual Request Volume | Avg. Versions per Project |
|---|---|---|---|
| Java (Maven) | 492k | 675B | 19 |
| JavaScript (npm) | 2.06M | 2.1T | 14 |

Open source enables significant reuse, easing initial development

But, use of open source is not zero cost...

Java application ~ 148 dependencies
Java project - 10 updates per year

... means application developers are tracking ~1500 dependency changes per year per project

* From sonatype, 8th annual State of the Software Supply Chain

# 01 Open Source

Software Supply Chain 2022*

| Ecosystem | Total Projects | Annual Request Volume | Avg. Versions per Project |
|---|---|---|---|
| Java (Maven) | 492k | 675B | 19 |
| JavaScript (npm) | 2.06M | 2.1T | 14 |

Open source enables significant reuse, easing initial development

But, use of open source is not zero cost…

Java application ~ 148 dependencies
Java project - 10 updates per year

… means application developers are tracking ~1500 dependency changes per year per project

* From sonatype, 8th annual State of the Software Supply Chain

01

# Open Source

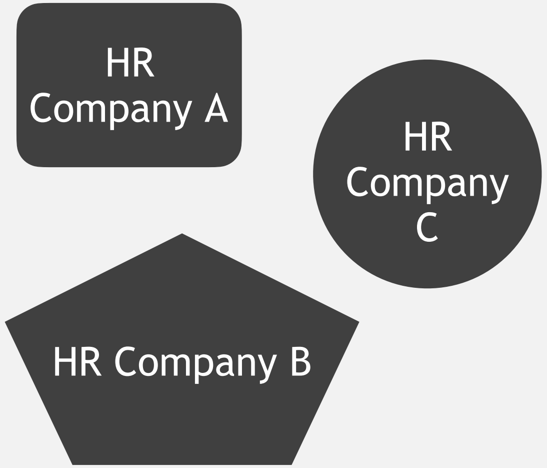
Open source reduces some development costs,
but incurs evolution costs

and as a result doesn't immediately provide an order of magnitude improvement

We'll revisit some costs later in the talk

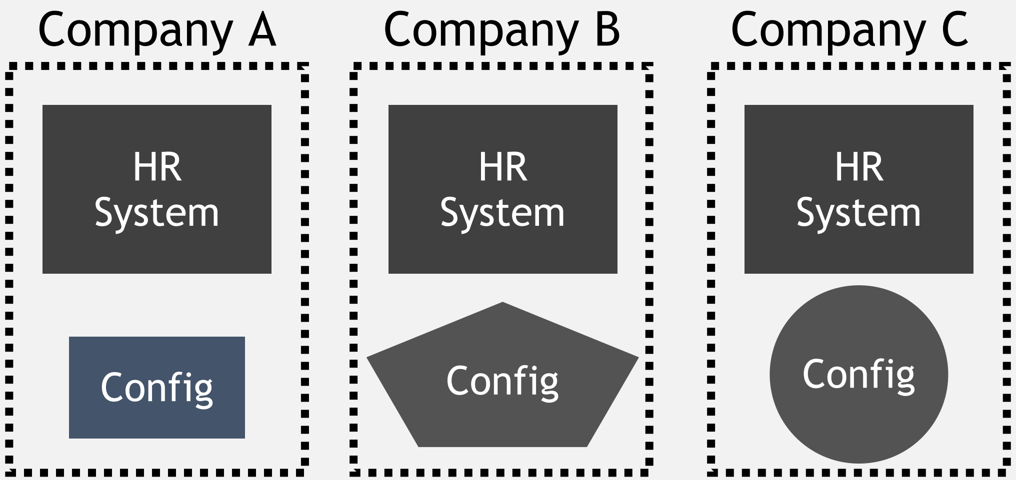
# 02 Cloud

1995




Unique on-premise
systems of similar functionality

2023




Same system with different
configurations and instances
for each company

# 02 Cloud

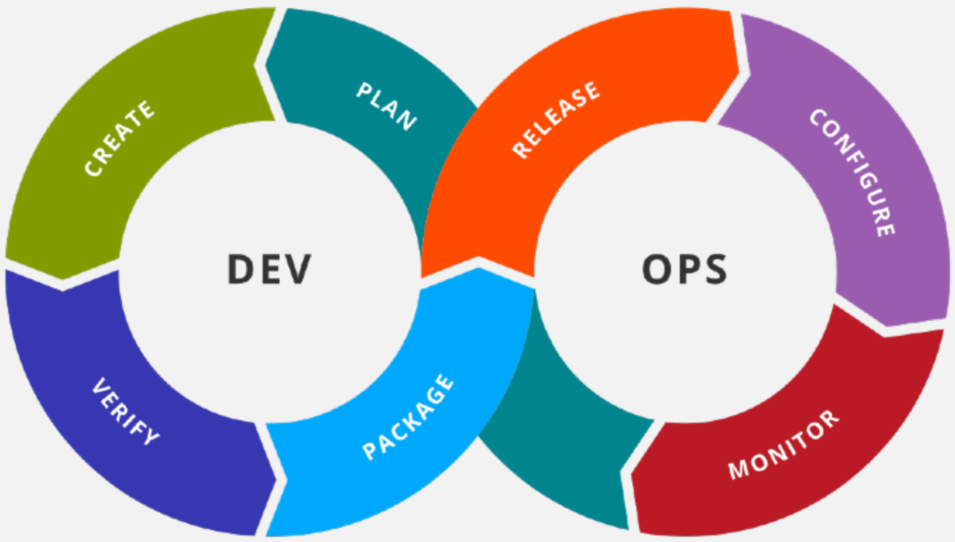
Use of the cloud has reduced development costs of similar systems

Organizations no longer need to build, but significant adoption and configuration costs

We’ll revisit some costs later in the talk

# 03 Automation

## DevOps




## Bots (Examples)

| | |
|---|---|
| Mergedroid | Automatically merge conflictive pull requests |
| Dependabot | Create pull requests to keep dependencies up-to-date |
| Danger | Automate team's code review conventions |

Picture from: https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg

## 03 Automation

Automation, in its many forms, has helped reduce friction in development and has helped speed up the release of software to users

Automation alone doesn’t help determine what system to build, how to design the system, etc.

04

# Generation

## SapFix: Automated End-to-End Repair at Scale

A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, A. Scott
Facebook Inc.

## Focus on solving bigger problems

Spend less time creating boilerplate and repetitive code patterns, and more time on what matters: building great software. Write a comment describing the logic you want and GitHub Copilot will immediately suggest code to implement the solution.

04

## Generation

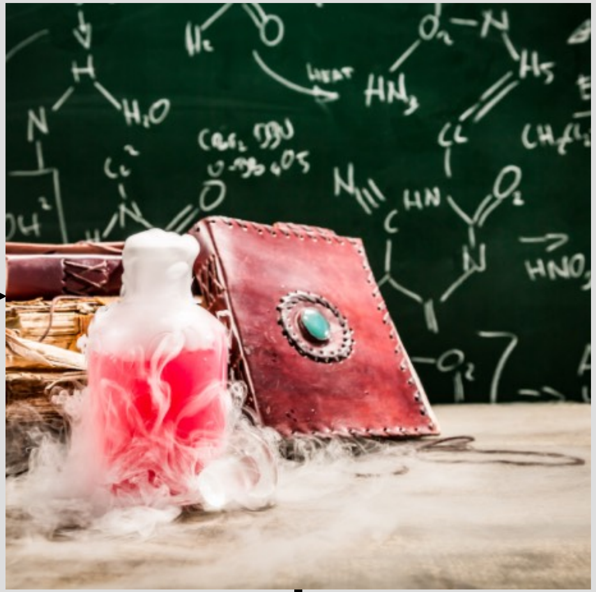
The generation possibilities with large language models for code, design, documentation, etc. are intriguing.

Will they significantly reduce effort of building and deploying systems or will we just build more complex systems?

# Some notable advances and ESSENTIAL COMPLEXITIES

| | | |
|---|---|---|
| 04 | Generation | MAYBE? |
| 03 | Automation | UNLIKELY? |
| 02 | Cloud | UNLIKELY? |
| 01 | Open source | UNLIKELY? |

“No silver bullet” recap



The last 25 years

**Essential complexities in 2023**


Research opportunities

Images not available for reuse

growing*

soft ware

* Brooks 1995

Images not available for reuse

# Expanding consideration of complexities from…

growing* to also using

soft ware

* Brooks 1995

Images not available for reuse


CONTEXT MATTERS
Images not available for reuse

# Context Matters

## Tacoma Narrows Bridge (1940)

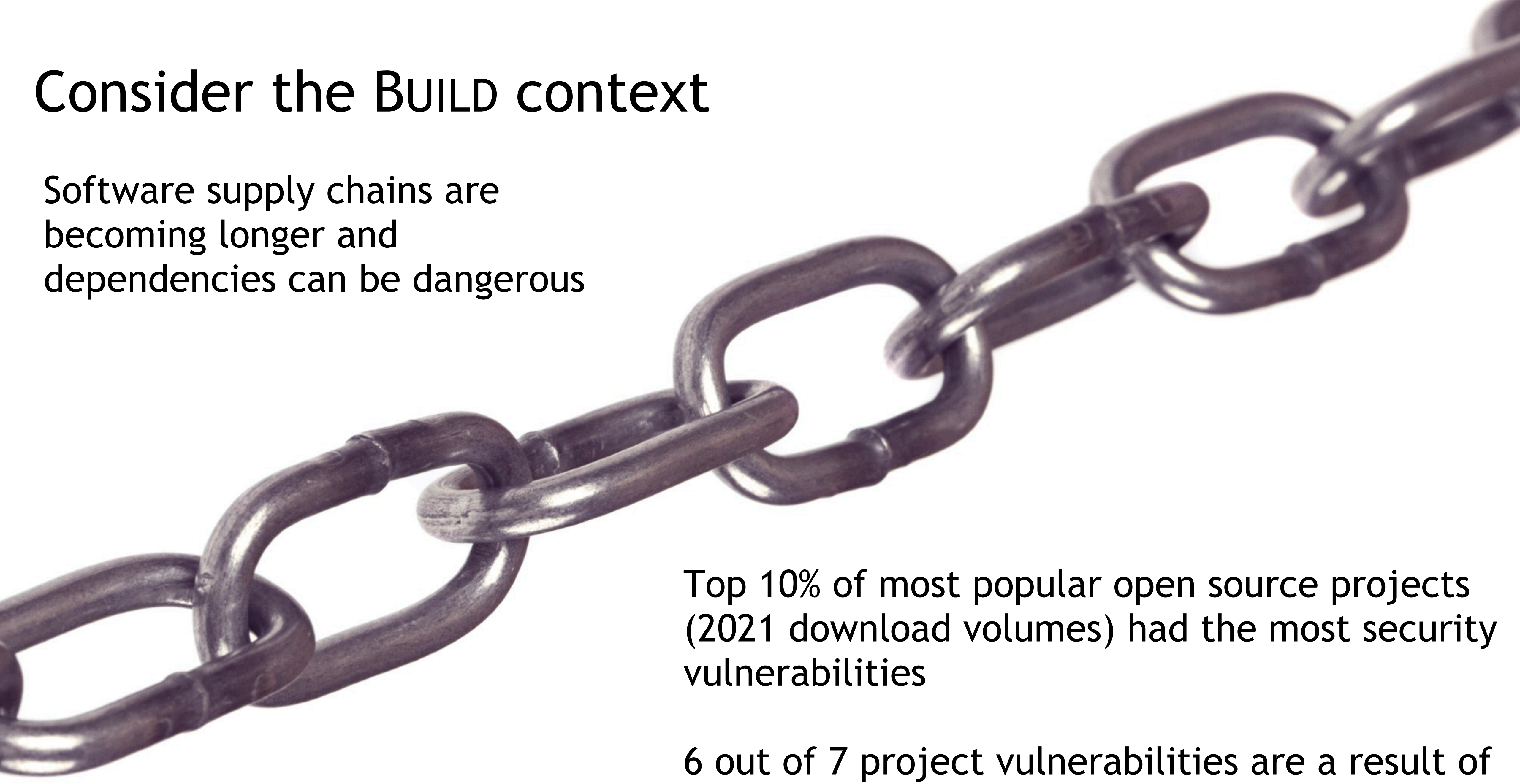
Clip from Prelinger Archives
(San Francisco)

Context Matters

Tacoma Narrows Bridge (1940)

Clip from Prelinger Archives (San Francisco)

# Consider the BUILD context



Software supply chains are becoming longer and dependencies can be dangerous

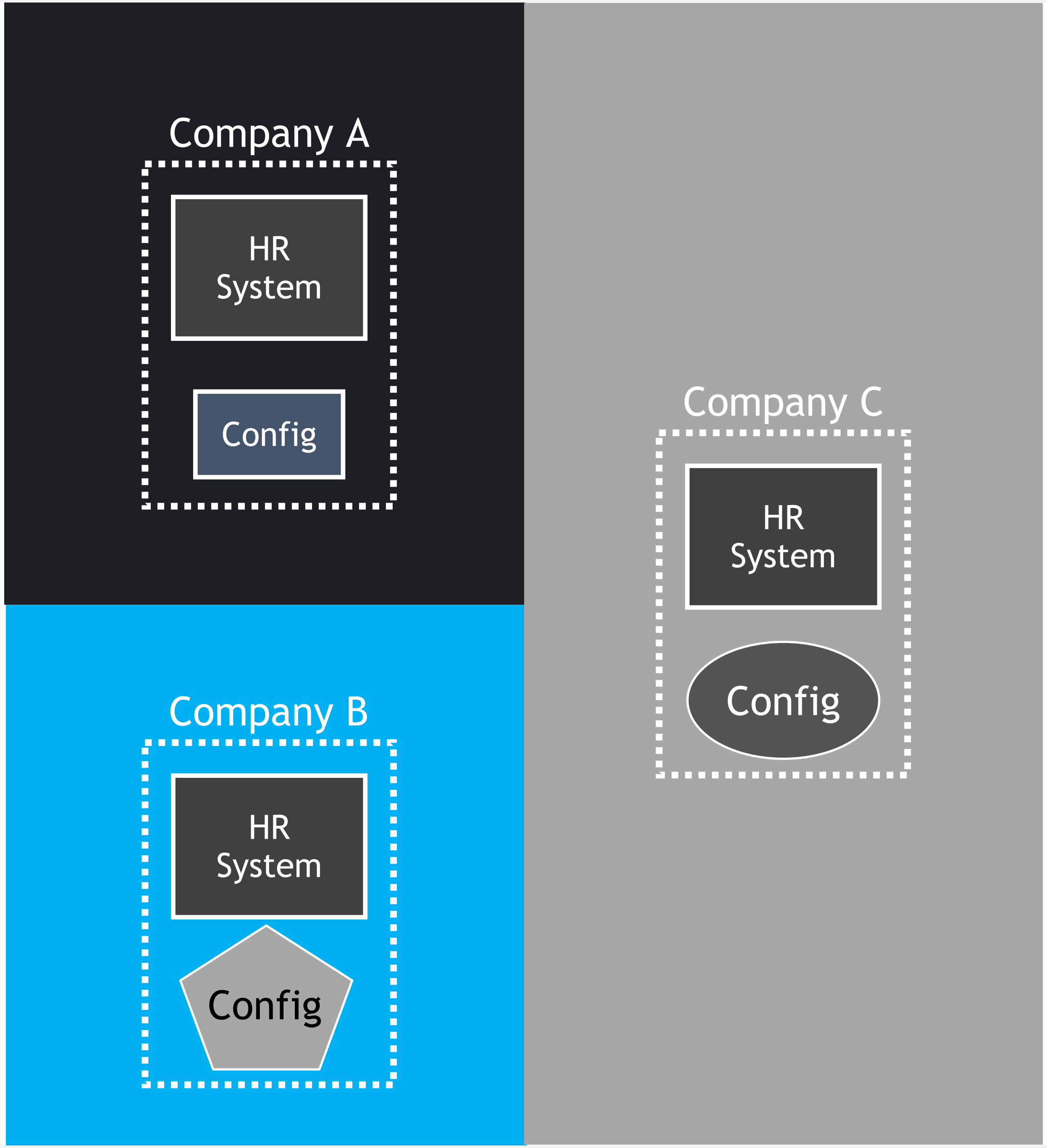
Top 10% of most popular open source projects (2021 download volumes) had the most security vulnerabilities

6 out of 7 project vulnerabilities are a result of transitive dependencies

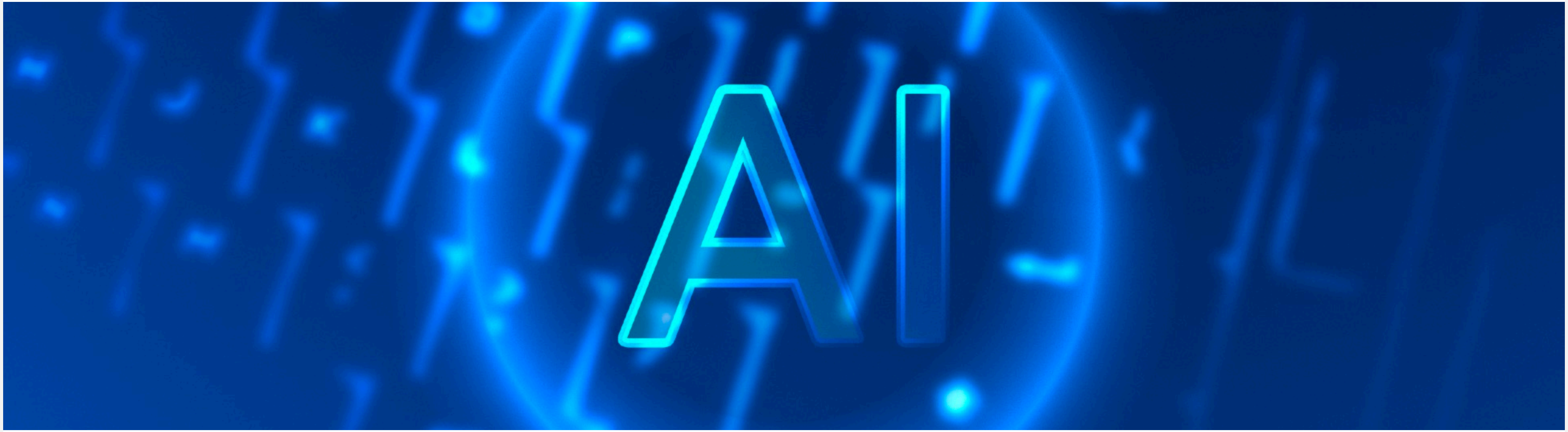
# Consider the Deployment context

Configuration files can significantly alter the behaviour of a cloud-deployed system

How do developers reason about, explain, grow, verify, etc. such systems once they are configured and in use?



# Consider the SOCIETAL context



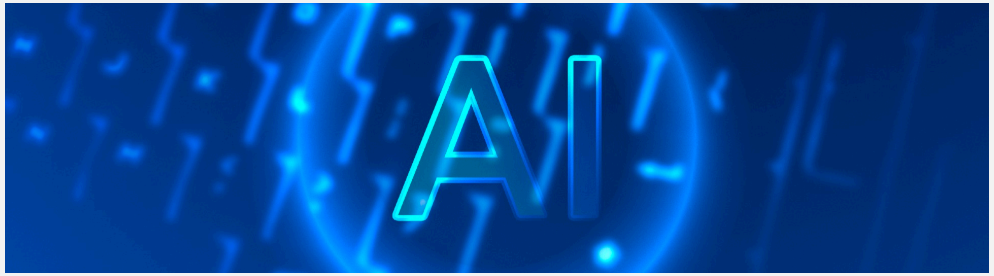

Embedding of AI techniques in software systems …

… introduces questions of fairness, non-determinism, … when the systems are in use

… makes various tasks of developing the software more challenging [Wan 2019]

Images not available for reuse

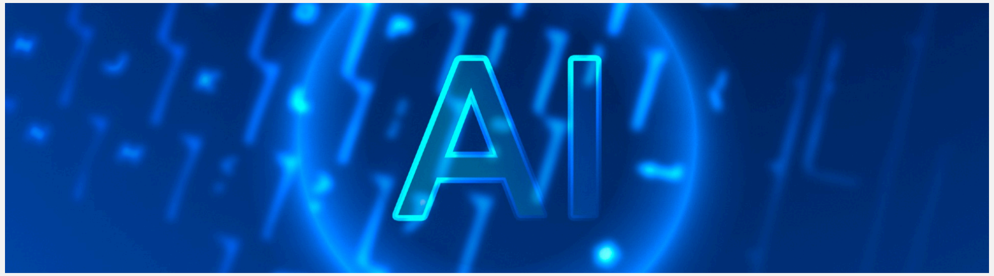
# Consider the SOCIETAL context




“Fixed” Data
(Deterministic)

Data-driven
(Non-deterministic)

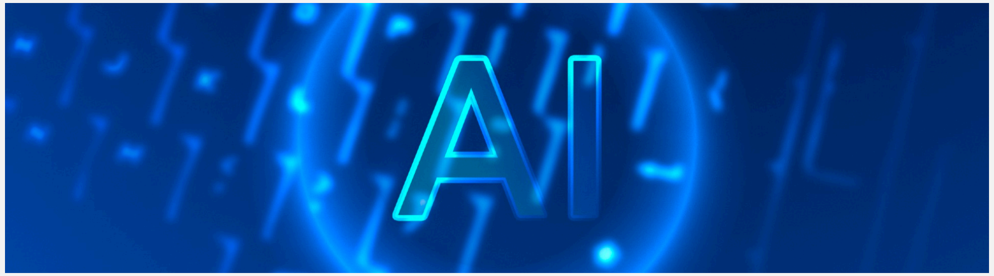
# Consider the SOCIETAL context




"Fixed" Data
(Deterministic)

Data-driven
(Non-deterministic)

# Consider the SOCIETAL context



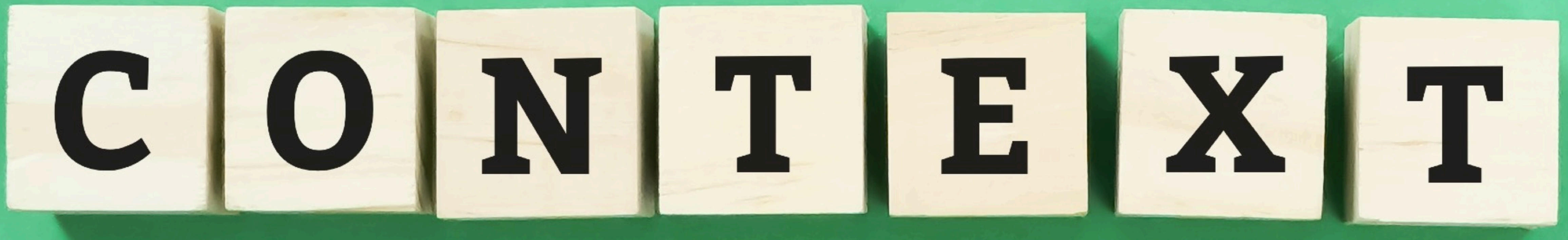

“Fixed” Data
(Deterministic)

Data-driven
(Non-deterministic)

ML vs. non-ML perspectives on development [Wan 2019] …

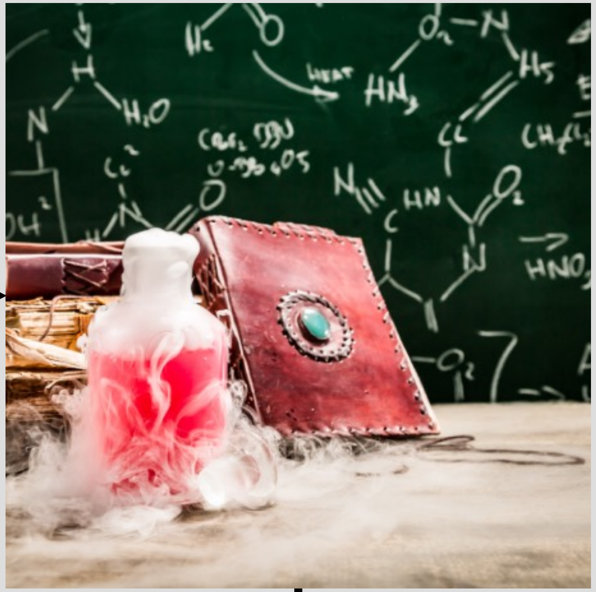
| Statement | | Likert Distributions ML (98) | Likert Distributions Non-ML (244) | Likert Distributions ML FTL (39) | Likert Distributions ML App (59) | Cliff’s Delta ML vs. Non-ML | Cliff’s Delta ML FTL vs. ML App | P-values ML vs. Non-ML | P-values ML FTL vs. ML App |
|---|---|---|---|---|---|---|---|---|---|
| Developing my software requires knowledge in math, information theory and statistics. | S24 | | | | | 0.45 | -0.[illegible]9 | ✔ .000 | .320 |
| Detailed design is time-consuming and conducted in an iterative way. | S7 | | | | | 0.32 | 0.18 | ✔ .000 | .271 |
| Requirements should consider predictable degradation in the performance of software. | S3 | | | | | 0.29 | 0.11 | ✔ .000 | .433 |
| It is easy to make an accurate plan for the development tasks of my software. | S29 | | | | | -0.32 | 0.03 | ✔ .000 | .779 |
| Data processing is important to the success of the whole development process. | S22 | | | | | 0.26 | -0.20 | ✔ .000 | .271 |
| Collecting testing dataset is labor intensive. | S15 | | | | | 0.27 | -0.26 | ✔ .000 | .188 |
| Developing my software requires frequent communications with the clients. | S31 | | | | | -0.29 | -0.14 | ✔ .000 | .577 |
| My software is tested by using automated testing tools. | S18 | | | | | 0.26 | 0.48 | ✘ .000 | ✔ .001 |
| Good testing results can guarantee the performance of my software in production. | S17 | | | | | -0.23 | 0.0[illegible] | ✔ .001 | .482 |
| Available data limit the capability my software. | S21 | | | | | 0.22 | -0.48 | ✔ .001 | ✔ .001 |
| Collecting requirements involve a large number of preliminary experiments. | S2 | | | | | 0.20 | 0.09 | ✔ .002 | .577 |

## Build, Deployment, Societal




Are these new essential complexities?

"No silver bullet" recap



The last 25 years

Essential complexities in 2023

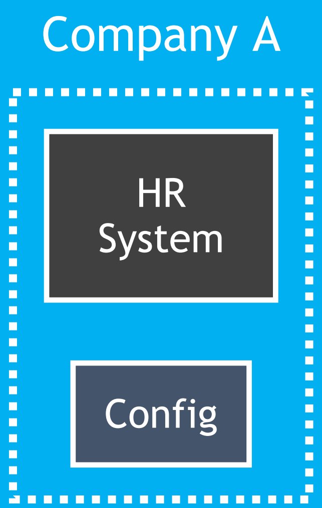
**Research opportunities**


Images not available for reuse

Need to consider whole software systems not just the parts

And the impact of the parts on the whole

Images not available for reuse




# For example … considering the whole

## What is the emergent behaviour for the HR system once configured?

How can functional testing be efficiently scaled across the entire configuration? (e.g., behavioural completeness)

How can a development team assure bounds of functionality in light of configurations? (e.g., behavioural consistency)

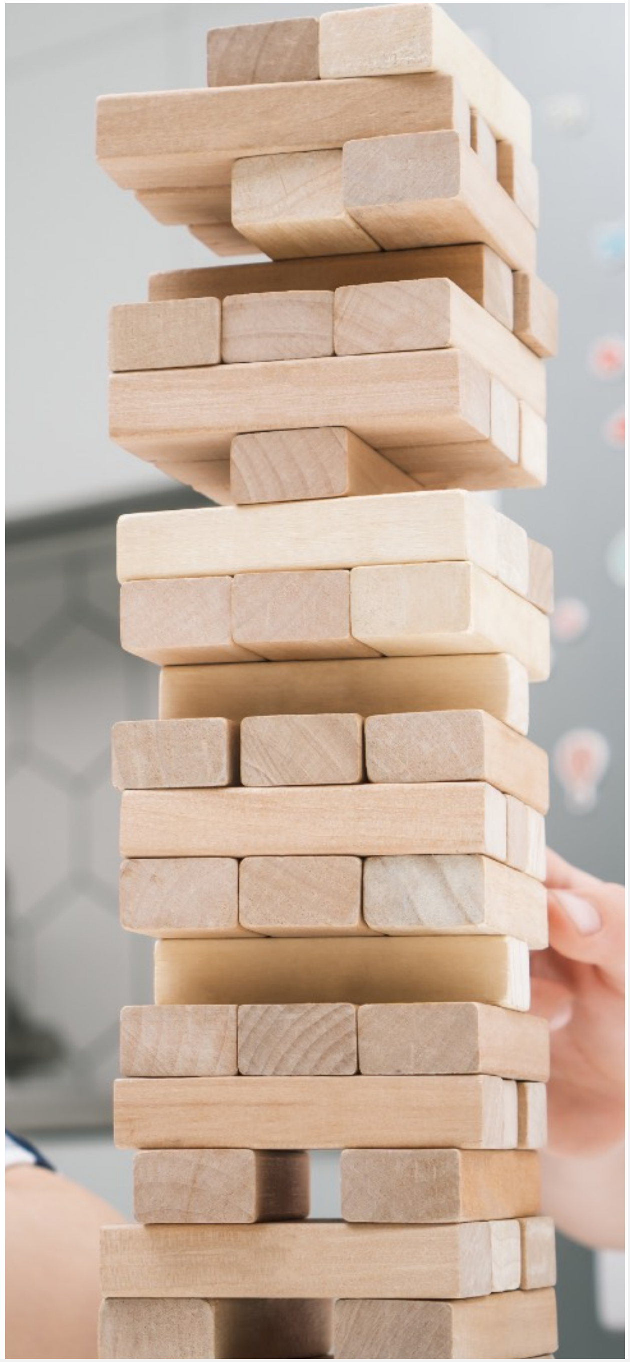
# For example … considering the impact of parts on the whole

How to estimate the costs of relying upon a software component, especially considering its transitive components?

How to efficiently update components as necessary (e.g., security updates)?

How to enhance components with checkable guarantees?

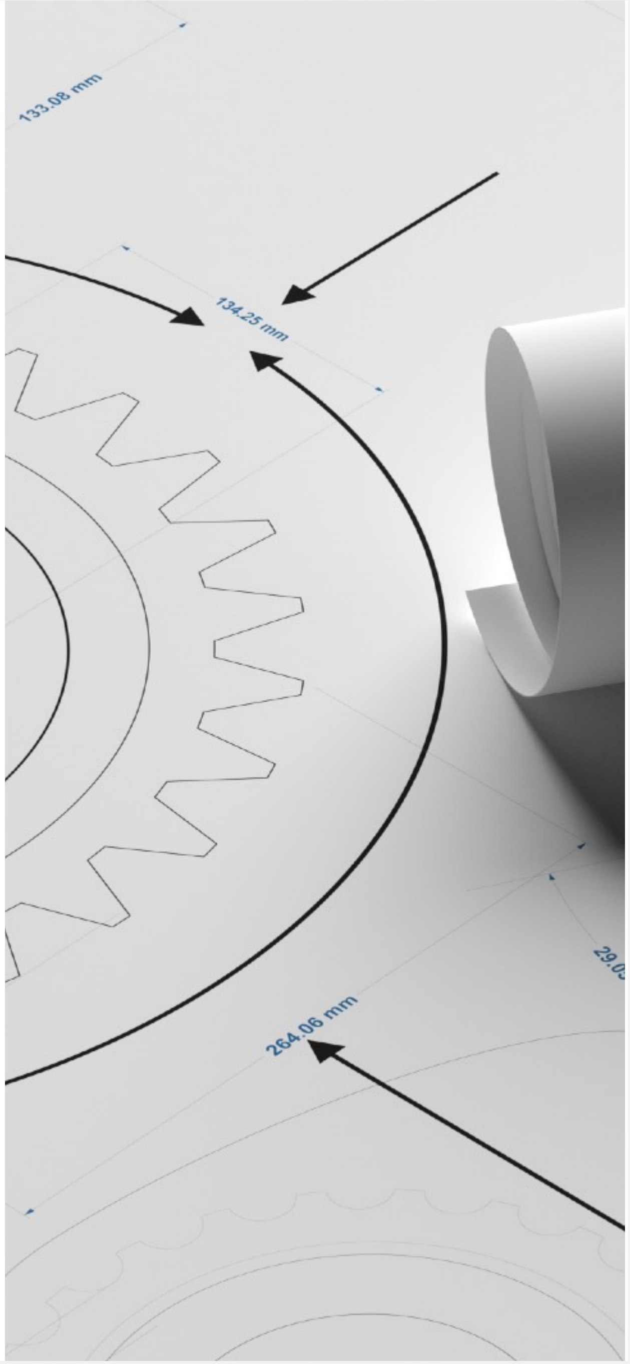
Images not available for reuse



# For example … considering the impact of parts on the whole

Are there design paradigms or patterns that can insulate more kinds of changes to parts of a system (e.g., beyond interface changes)?

Are there designs that are evolve more gracefully with changes in the environment in which the system must run?



# How can we move software engineering research towards these questions?

More study of longitudinal development

More study of deployed systems at scale

More integration of research results to solve bigger problems

Academic community (and funding agencies) need to accept different forms of impact as excellent research (e.g., long-term case studies, integrative results)

Society needs to see value in studying systems at scale

# Thank You

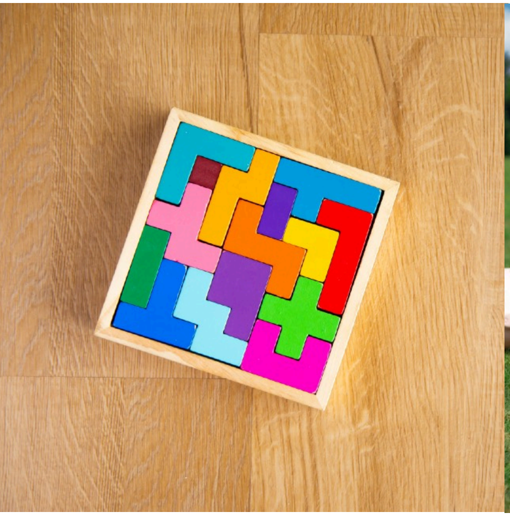
To the many talented students (undergraduate and graduate), post-doctoral fellows and colleagues that I have been fortunate to work with

To NSERC for long-term funding

To my co-founders and colleagues at Tasktop Technologies for an amazing journey full of learnings

And the conference organizers for this invitation

# Software development has essential complexities





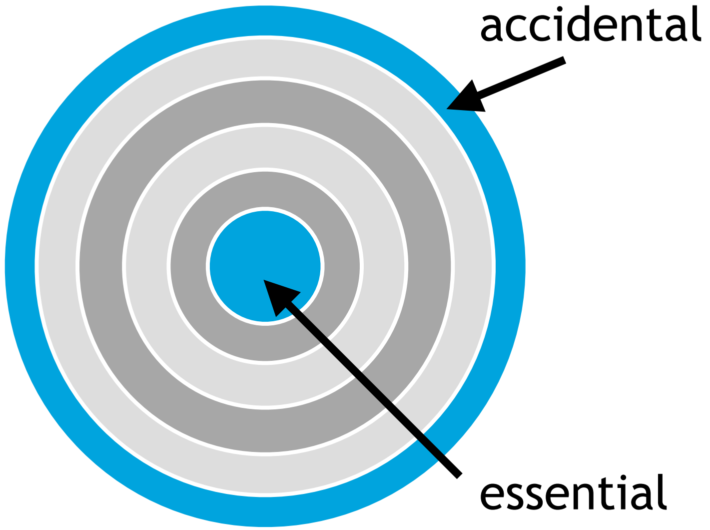
When viewed over time include contextual (build, deployment, use) essential complexities

Images not available for reuse

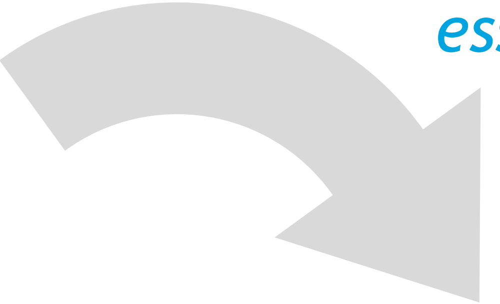
In addition to continuing focused technology development and laboratory study, we need to study more systems in-situ and at scale to better understand and address essential complexities

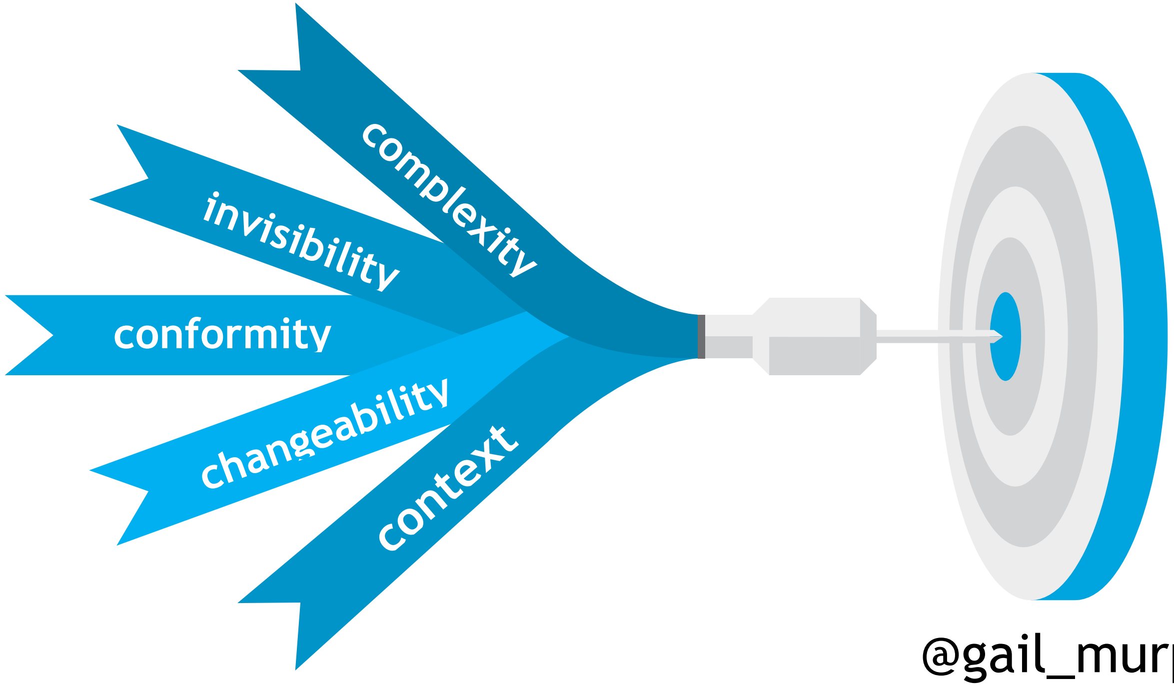
Need to re-consider our academic and funding criteria and assessments

# Take-aways




Move from foci on *accidental* complexities to more study about the *essential* complexities of growing software






Consider more…

holistic, longitudinal and interdisciplinary study of software in-situ and at scale…

which has implications for funding and research assessments


@gail_murphy
gail_murphy@social.sigsoft.org