Tackling the ever changing essential complexities of engineering software

Gail C. Murphy
University of British Columbia

@gail_murphy
gail_murphy@social.sigsoft.org
Bridges

1500 BCE
Holzbrücke
Rappersweil-Hurden
(wooden timber)

Arkadiko
1300 BCE

Zhaozhou
600 CE

Wetherby
1200 CE

Akashi Kaikyo
1998 CE
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
Images not available for reuse

**Bridges**

- Holzbrücke, Rappersweil-Hurden (wooden timber) in 1500 BCE
- Arkadiko in 1300 BCE
- Zhaozhou in 600 CE
- Wetherby in 1200 CE
- Akashi Kaikyo in 1998 CE

~3500 years

**Software**

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

~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.
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.
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
“No silver bullet” recap

The last 25 years
“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
“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.
“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.”
“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.”
“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.”
Brook’s Essential Complexities (1986)

**Complexity**

No two parts are alike  
Many parts needed

**Conformity**

Software most conformable  
Complexity from conforming
Changeability

Software is constantly subject to change and is infinitely changeable

Invisibility

“Software is invisible and unvisualizable”
Improvements have addressed accidental (incidental) complexities

Brook’s Essential Complexities Remain (1995)
“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

- Open source
- Cloud
- Automation
- Generation
Open Source

Software Supply Chain 2022*

<table>
<thead>
<tr>
<th>Ecosystem</th>
<th>Total Projects</th>
<th>Annual Request Volume</th>
<th>Avg. Versions per Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java (Maven)</td>
<td>492k</td>
<td>675B</td>
<td>19</td>
</tr>
<tr>
<td>JavaScript (npm)</td>
<td>2.06M</td>
<td>2.1T</td>
<td>14</td>
</tr>
</tbody>
</table>

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
Open Source

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

---

Software Supply Chain 2022*

<table>
<thead>
<tr>
<th>Ecosystem</th>
<th>Total Projects</th>
<th>Annual Request Volume</th>
<th>Avg. Versions per Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java (Maven)</td>
<td>492k</td>
<td>675B</td>
<td>19</td>
</tr>
<tr>
<td>JavaScript (npm)</td>
<td>2.06M</td>
<td>2.1T</td>
<td>14</td>
</tr>
</tbody>
</table>

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

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

Software Supply Chain 2022*

<table>
<thead>
<tr>
<th>Ecosystem</th>
<th>Total Projects</th>
<th>Annual Request Volume</th>
<th>Avg. Versions per Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java (Maven)</td>
<td>492k</td>
<td>675B</td>
<td>19</td>
</tr>
<tr>
<td>JavaScript (npm)</td>
<td>2.06M</td>
<td>2.1T</td>
<td>14</td>
</tr>
</tbody>
</table>

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

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
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.
Unique on-premise systems of similar functionality

Same system with different configurations and instances for each company
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.
**Automation**

**DevOps**

- **CREATE**
- **PLAN**
- **RELEASE**
- **CONFIGURE**
- **MONITOR**

**OPS**

- **PACKAGE**

**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
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.
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.
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

Open source

Cloud

Automation

Generation

ESSENTIAL COMPLEXITIES

Maybe?

Unlikely?
“No silver bullet” recap

The last 25 years

Research opportunities

Essential complexities in 2023
growing

software

* Brooks 1995
Expanding consideration of complexities from... growing* to also using software

* Brooks 1995
CONTEXT MATTERS
Context Matters

Tacoma Narrows Bridge (1940)
Context Matters

Tacoma Narrows Bridge (1940)
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]
Consider the SOCIETAL context

“Fixed” Data (Deterministic)  Data-driven (Non-deterministic)
Consider the **SOCIAL** 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] ...**

<table>
<thead>
<tr>
<th>Statement</th>
<th>Likert Distributions</th>
<th>Cliff’s Delta ML vs. Non-ML</th>
<th>Cliff’s Delta ML FTL vs. Non-ML</th>
<th>P-values ML vs. Non-ML</th>
<th>P-values ML FTL vs. Non-ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing my software requires knowledge in math, information theory and statistics.</td>
<td>S24</td>
<td>0.45</td>
<td>0.59</td>
<td>0.000</td>
<td>0.320</td>
</tr>
<tr>
<td>Detailed design is time-consuming and conducted in an iterative way.</td>
<td>S7</td>
<td>0.32</td>
<td>0.38</td>
<td>0.000</td>
<td>0.271</td>
</tr>
<tr>
<td>Requirements should consider predictable degradation in the performance of software.</td>
<td>S3</td>
<td>0.29</td>
<td>0.11</td>
<td>0.000</td>
<td>0.433</td>
</tr>
<tr>
<td>It is easy to make an accurate plan for the development tasks of my software.</td>
<td>S29</td>
<td>-0.32</td>
<td>-0.11</td>
<td>0.000</td>
<td>0.779</td>
</tr>
<tr>
<td>Data processing is important to the success of the whole development process.</td>
<td>S22</td>
<td>0.26</td>
<td>-0.20</td>
<td>0.000</td>
<td>0.271</td>
</tr>
<tr>
<td>Collecting testing dataset is labor intensive.</td>
<td>S15</td>
<td>0.27</td>
<td>-0.16</td>
<td>0.000</td>
<td>0.188</td>
</tr>
<tr>
<td>Developing my software requires frequent communications with the clients.</td>
<td>S31</td>
<td>-0.29</td>
<td>-0.14</td>
<td>0.000</td>
<td>0.577</td>
</tr>
<tr>
<td>My software is tested by using automated testing tools.</td>
<td>S18</td>
<td>0.26</td>
<td>0.18</td>
<td>0.000</td>
<td>0.201</td>
</tr>
<tr>
<td>Good testing results can guarantee the performance of my software in production.</td>
<td>S17</td>
<td>-0.23</td>
<td>0.02</td>
<td>0.001</td>
<td>0.482</td>
</tr>
<tr>
<td>Available data limit the capability my software.</td>
<td>S21</td>
<td>0.22</td>
<td>-0.48</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>Collecting requirements involve a large number of people or in minimize.</td>
<td>S1</td>
<td></td>
<td></td>
<td>0.001</td>
<td>0.001</td>
</tr>
</tbody>
</table>
Build, Deployment, Societal

Context

Are these new essential complexities?
"No silver bullet" recap

The last 25 years

Research opportunities

Essential complexities in 2023

Images not available for reuse
Need to consider whole software systems not just the parts

And the impact of the parts on the whole
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
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