AI-Augmented Software Engineering: Opportunities and Implications

CIBSE 2023
April 26, 2023

Ipek Ozkaya
ozkaya@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

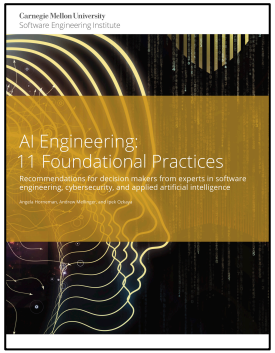
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

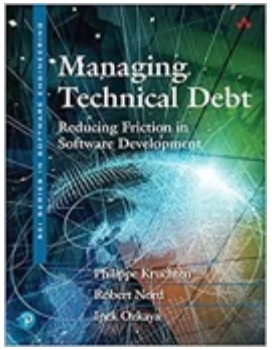
DM23-0413

About me

Carnegie Mellon University

PhD in Computational Design from CMU
Technical Director, Engineering Intelligent Software Systems at the SEI





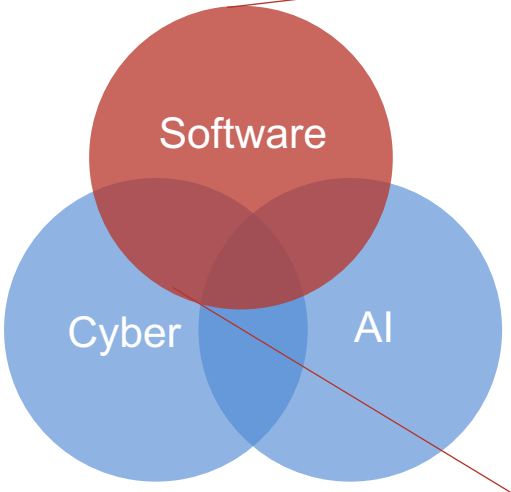
Body of work at the intersection of architecture design, analysis, and tradeoffs

IEEE Software Magazine
Editor-in-Chief

Istanbul, Turkey

Pittsburgh, PA USA

Software Engineering at the SEI:
Rapidly Deploying Software Innovations with Confidence



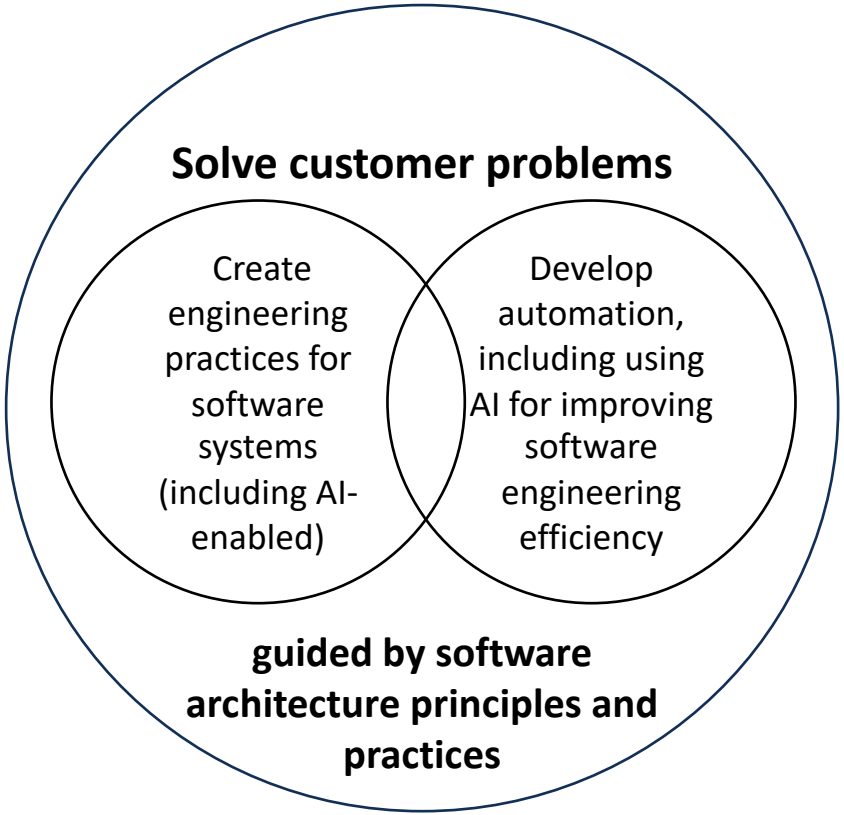
SEI Strategic Framework

Exploit emerging technologies

- Engineering Intelligent Software Systems
- Enabling Mission Capability at Scale
- Assuring Cyber-Physical Systems
- Continuous Deployment of Capability
- Transforming Software Acquisition Policy and Practice

Curate & analyze data for decision making

Engineering Intelligent Software Systems – 1



Develop and apply range of techniques and practices applicable at different points in the software development lifecycle.

- Domains of expertise include IT, C2, tactical, avionics, edge, and health informatics
- Technology expertise includes IoT, big data, digital twin, cloud, and machine learning

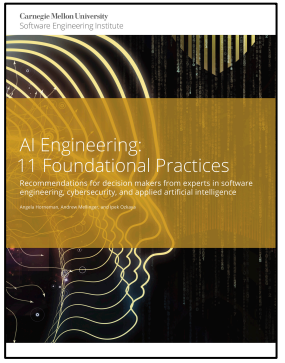
Engineering Intelligent Software Systems – 2

Principles, practices and tools developed by the SEI are widely used for architecting software systems.

We address challenges of incorporating emerging and next generation technical advancements building on our proven techniques and ongoing research, prototyping and data analysis.

The SEI Pearson Addison-Wesley Series on Software Architecture

K. A. Pitstick, M. Novakouski, G. Lewis, I. Ozkaya *Computing at the Edge: Challenges and Priorities for Software Engineering and AI*. CMU SEI, 2021.



A. Horneman, A. Mellinger, I. Ozkaya. *AI Engineering: 11 Foundational Practices*. CMU SEI, 2019.

AI-Augmented Software Development – The Landscape

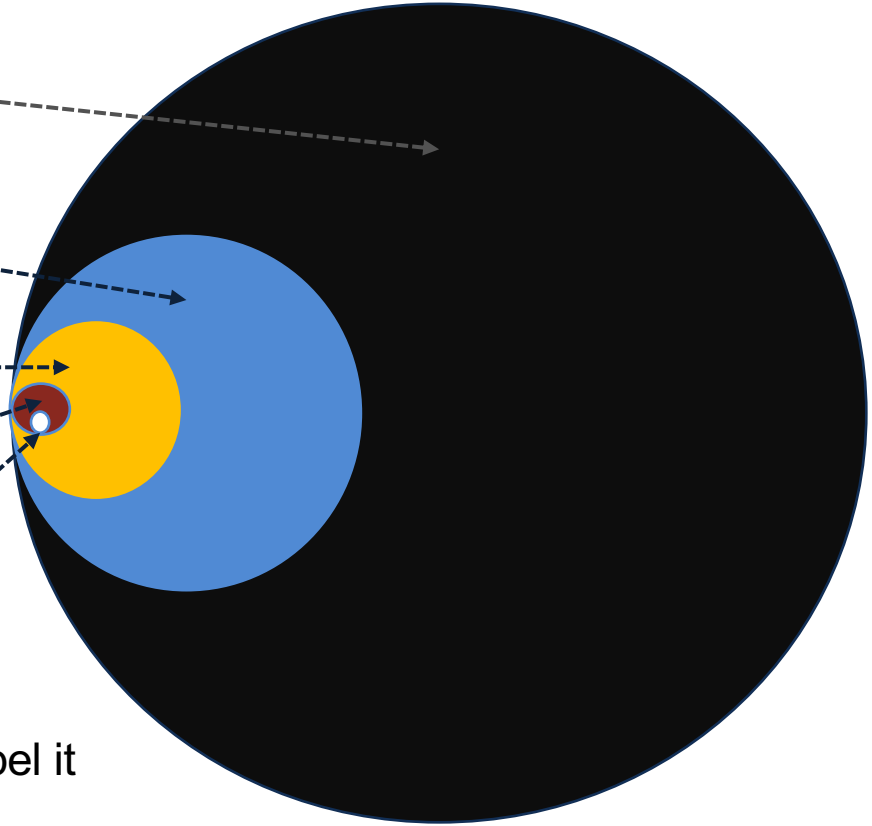
People who talk about it

People who understand it

People who research it

People who implement solutions for it

People who develop next generation AI algorithms and hardware solutions to propel it

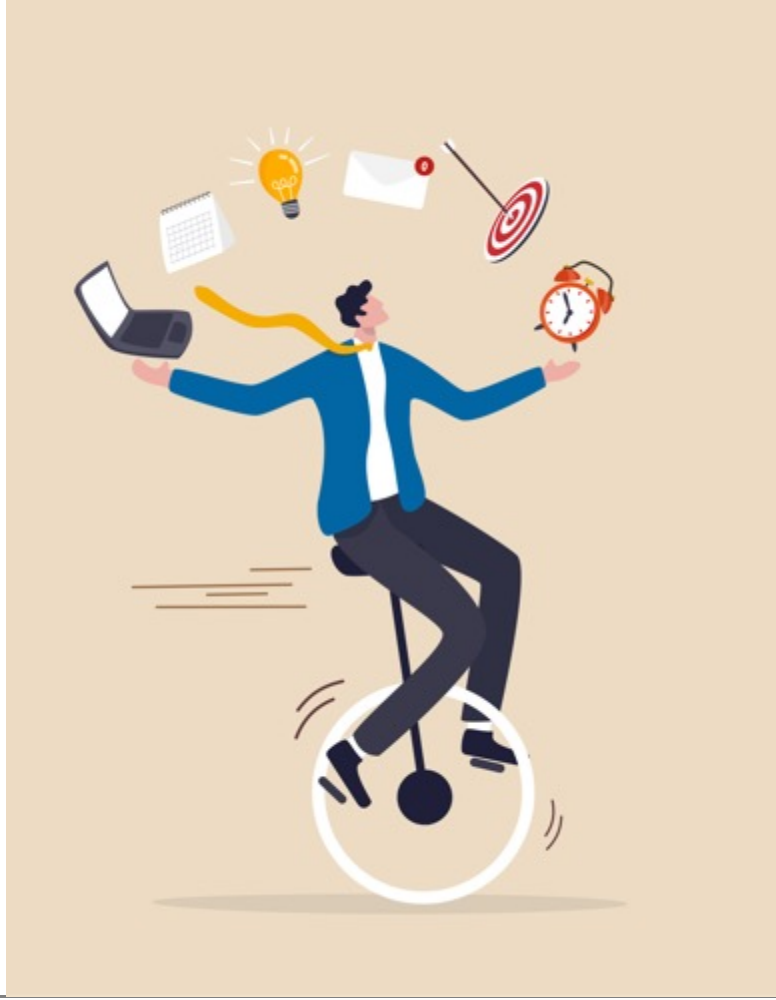


BLUF
(Bottom Line Up Front)

Improving **developer productivity**, consequently system quality, has been a key concern in software engineering for decades.

A focus on improved automation, including AI-augmented tools (*your favorite generative AI tool too*), **is neither new, nor novel.**

The opportunity (and challenge) for software engineering community is to discover **whether the fast pace improvements in AI-assistants change how we engage with and orchestrate software development activities.**

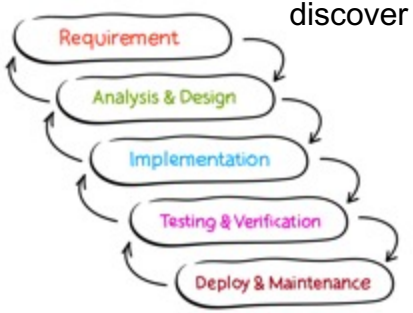


A fool with a tool is still a fool!

Grady Booch

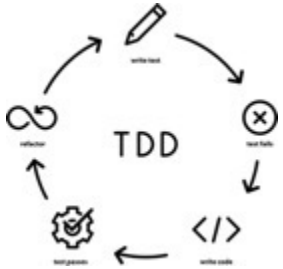
Software Development Lifecycle Evolution

We introduce delays as we discover issues of each phase



We get stuck when deploying to operations

We find out mistakes too late



Success: Well-defined software engineering tasks

Challenge: No feedback loops, hence delays

Automation efforts:

- Modeling tools to support design and conformance
- Requirement traceability tools for consistency checking and developing the right thing

Success: Improved communication and delivery of relevant functionality

Challenge: Chaos and unintended rework

Automation efforts:

- Workflow management tools
- Improved code analyzers
- Progress on our understanding of developer behavior

Success: Improved delivery tempo

Challenge: Reduce hand-overs with automation

Automation efforts:

- Automated code review
- Automated testing
- Infrastructure as code
- Configuration as code
- Program repair

Yet still….

Software teams cannot get ahead of:

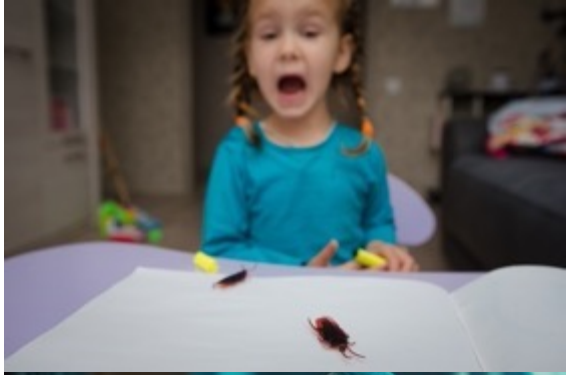
- Defects, vulnerabilities, and technical debt issues
- Privacy and security leaks
- Complexity and rework rooted cost overruns

In mission critical settings in particular, trust and assurance cannot we guaranteed.

Continuous software engineering and evolution are myths.

Brownfield development and software composition *(design and architecture)* are not supported by tools.

Organizations continue to lack skilled software engineers.




Have no fear, AI is here?

Can AI-augmented Software Development Accomplish….

10x reduction in resource needs and error rates.

Support at higher levels of abstraction to assist developers manage ripple effects in complex systems.

Making resources available to cognitively complex activities.

Reducing the need for extensive testing and analysis.

Guaranteed security, performance, conformance to quality standards and intended architectures.

TRUST!

Architecting the Future of Software Engineering

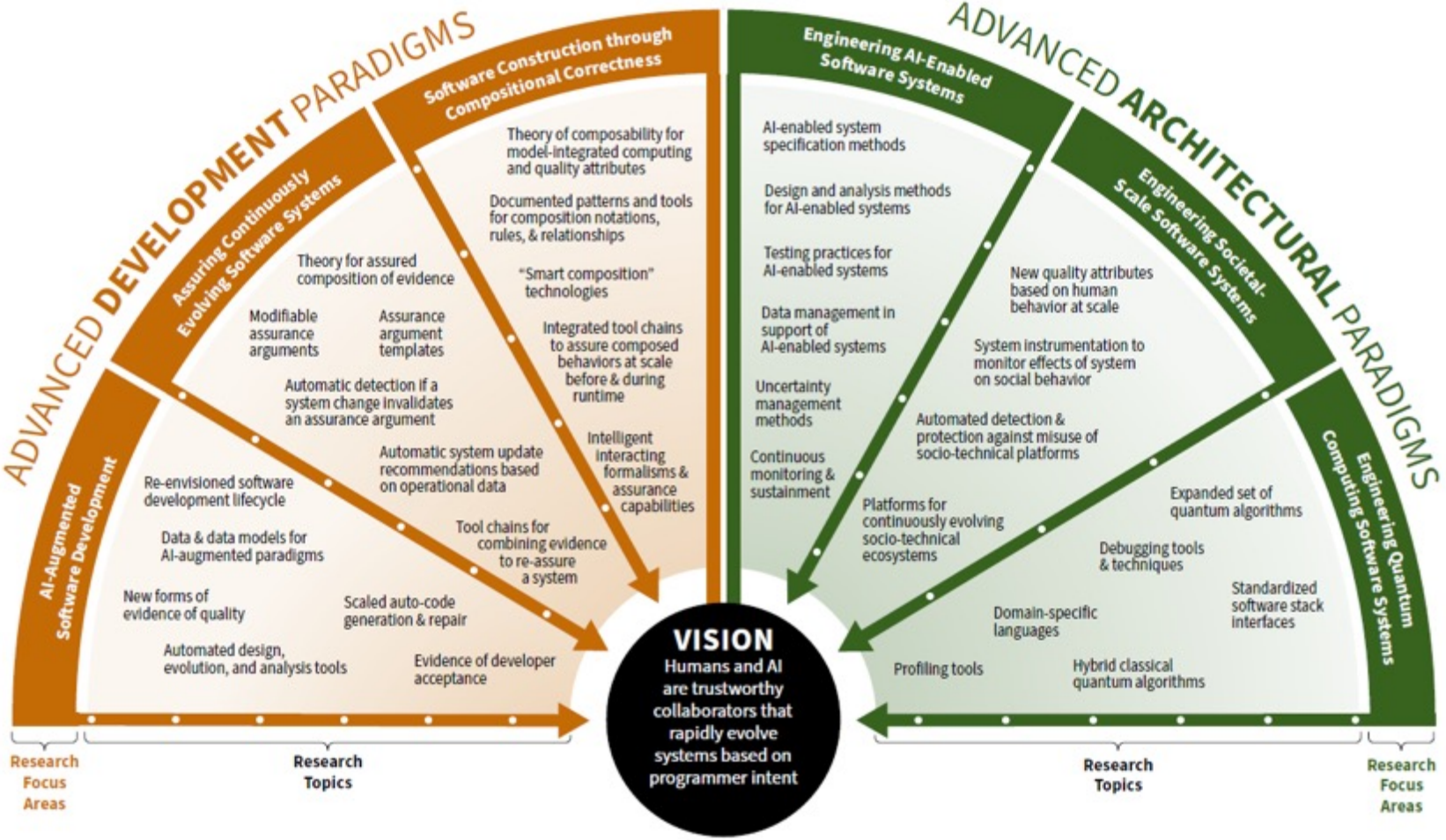
A National Agenda for Software Engineering Research & Development

Anita Carleton, et. al

November 2021

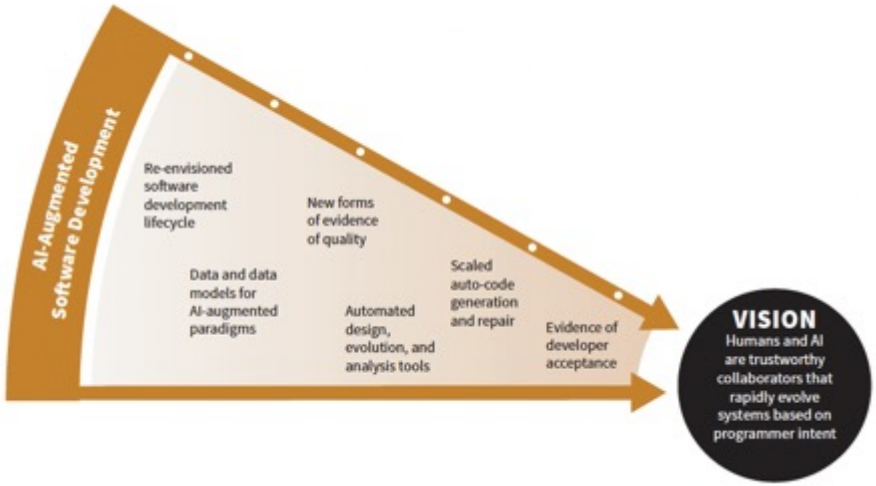
https://www.sei.cmu.edu/go/national-agenda

Software Engineering Research Roadmap (10-15 Year Horizon)



AI-Augmented Software Development

Do better versus Do differently?



Use of **AI-based (*and other*) automated tools** to improve the efficiency of software engineers and reduce their cognitive load, **shifting the attention** of humans to the **conceptual tasks** that computers are not good at and **eliminating human error from** tasks where computers can help.

A new, multimodal human-computer “partnership” model:

- Intern who I don't entirely trust, but who does save me a lot of time
- Bot that does things for me
- Partner that advises me

Innovations at the intersection of **DO BETTER** versus **DO DIFFERENTLY**

Do better – 1

Test automation: design test cases, localise and triage crashes, monitor their fixes

- *Example:* N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, I. Zorin: *Deploying Search Based Software Engineering with Sapienz at Facebook.* SSBSE 2018: 3-45
- *AI Approach:* Multi-objective search

Defect prediction: Develop more accurate predictors with less data

- Example: NC Shrikanth, T Menzies: *The Early Bird Catches the Worm: Better Early Life Cycle Defect Predictors*. https://arxiv.org/abs/2105.11082 2021.
- *AI approach:* early life cycle within-project transfer learning

Requirement traceability: Integrate information that is currently disconnected

- *Example:* J. Lin, Y. Liu, Q. Zeng, M. Jiang, J. Cleland-Huang: *Traceability Transformed: Generating more Accurate Links with Pre-Trained BERT Models*. ICSE 2021:
- *AI approach:* NLP, specifically BERT (Bidirectional Encoder Representations from Transformers)

Do better – 2

Code review: automatic, flexible, and adaptive code analysis to recommend reviews

- *Example:* Anshul Gupta, Neel Sundaresan, *Intelligent code reviews using deep learning* KDD'18 Deep Learning Day, August 2018, London, UK
- *AI Approach:* Deep neural networks, NLP

Program repair: automatically fixing instances of common bugs learning from past fixes

- *Example*: J. Bader, A. Scott, M. Pradel, S. Chandra. 2019. *Getafix: learning to fix bugs automatically.* Proc. ACM Program. Lang. 3, OOPSLA
- *AI approach:* hierarchical clustering to learn past fix patterns

Auto code completion: Complete code snippets as developers implement code

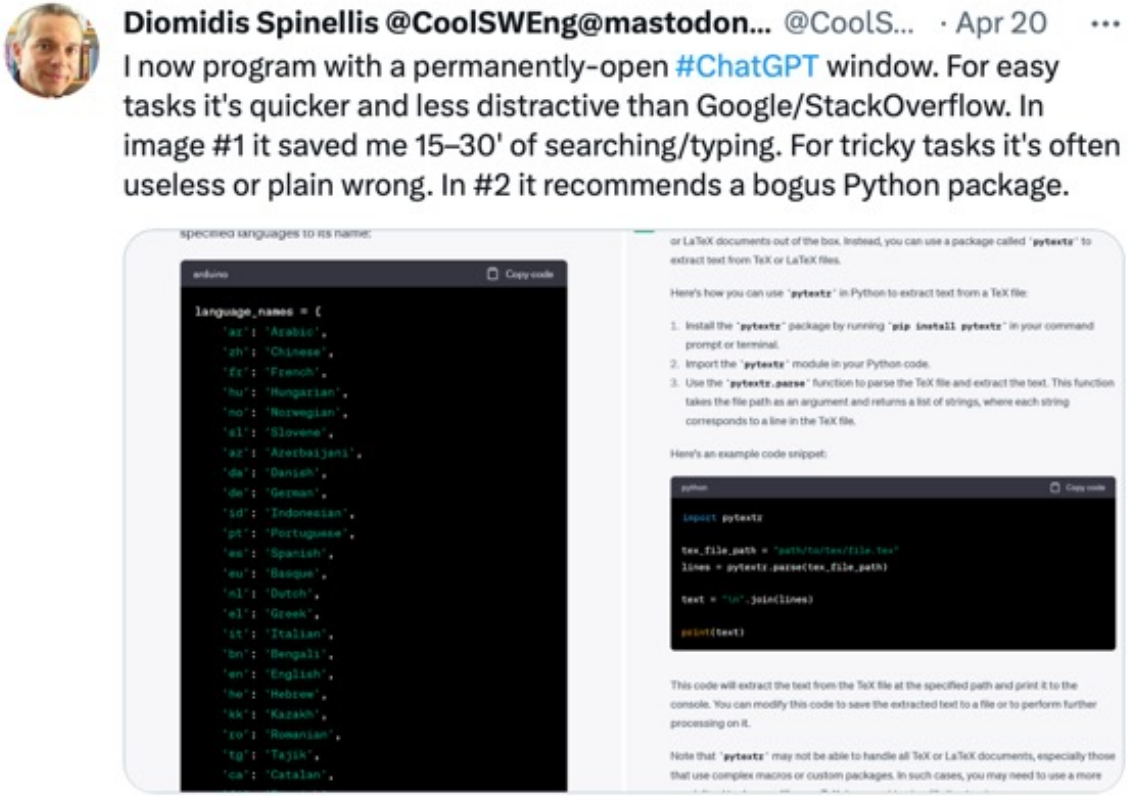
- *Example:* C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit. 2023. *Taking Flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools*. Queue 20, 6, (November/December)
- *AI approach:* Generative AI, large language models

Do differently
Generative AI – Existential threat or snake oil

- Will entry to software engineering activities be easier?
- Will some activities be obsolete within 5 years (democratize activities)?
- Will correctness and efficiency increase?
- Will some roles disappear?

- Garbage in garbage out
- Limitations on training data and domain applications
- Dependency on expertise to validate correctness
- Uncertainty in recommendations
- Privacy, security, bias concerns
- Creativity, ownership, and innovation concerns

Generative AI – Existential threat or snake oil



Diomidis Spinellis @CoolSWEng@mastodon... @CoolS... · Apr 20

I now program with a permanently-open #ChatGPT window. For easy tasks it's quicker and less distractive than Google/StackOverflow. In image #1 it saved me 15–30' of searching/typing. For tricky tasks it's often useless or plain wrong. In #2 it recommends a bogus Python package.

specified languages to its name:

```
language_names = {
    'ar': 'Arabic',
    'zh': 'Chinese',
    'fr': 'French',
    'hu': 'Hungarian',
    'no': 'Norwegian',
    'sl': 'Slovene',
    'az': 'Azerbaijani',
    'da': 'Danish',
    'de': 'German',
    'id': 'Indonesian',
    'pt': 'Portuguese',
    'es': 'Spanish',
    'eu': 'Basque',
    'nl': 'Dutch',
    'el': 'Greek',
    'it': 'Italian',
    'bn': 'Bengali',
    'en': 'English',
    'he': 'Hebrew',
    'kk': 'Kazakh',
    'ro': 'Romanian',
    'tg': 'Tajik',
    'ca': 'Catalan',
```

or LaTeX documents out of the box. Instead, you can use a package called `pytextx` to extract text from TeX or LaTeX files.

Here's how you can use `pytextx` in Python to extract text from a TeX file:

1. Install the `pytextx` package by running `pip install pytextx` in your command prompt or terminal.
2. Import the `pytextx` module in your Python code.
3. Use the `pytextx.parse` function to parse the TeX file and extract the text. This function takes the file path as an argument and returns a list of strings, where each string corresponds to a line in the TeX file.

Here's an example code snippet:

```
import pytextx

tex_file_path = "path/to/tex/file.tex"
lines = pytextx.parse(tex_file_path)

text = "\n".join(lines)

print(text)
```

This code will extract the text from the TeX file at the specified path and print it to the console. You can modify this code to save the extracted text to a file or to perform further processing on it.

Note that `pytextx` may not be able to handle all TeX or LaTeX documents, especially those that use complex macros or custom packages. In such cases, you may need to use a more

Maybe both?

Confusions between an LLM tool versus an LLM-based SE tool

LLMs can also answer some implementation questions

ChatGPT (by Open AI)
- ChatGPT released v4 is much more effect in many areas
- Copilot uses the same underlying model

LLaMA (by Meta)
- Small to larger versions of the model are available

Alpaca (by Stanford)
- Stanford copies the ChatGPT AI for less than $600 and approach can be replicated to train the model

AlexaTM (by Amazon)
- Amazon SageMaker enables training LLMs

KOSMOS-1 (by Microsoft)
- A multimodel Large language model (MLLM) that looks at more than language

Bard (by Google)
- ~~Still "learning to code"~~
- As of 5 days ago, can now code

LLM-based SE tools

Github Copilot X build on OpenAI GPT-4
- brings chat and voice interfaces, support pull requests, answer questions on docs
- they position their vision as "generative AI represents the future of software development"

CodiumAI using LLMs to recommend tests

Tabnine to assist developers with code completion using LLMs

Synk Code to find and automatically fix vulnerabilities in code, open source dependencies, containers, and infrastructure as code using text-to-text

What can we Do differently…

Do tasks developers aren't able to do today (e.g., leverage new data to integrate new conformance checks or generate new tests).

Scale and optimize beyond what developers already can do (e.g., consider more alternative design options)

Change the scope of activities (e.g. what are the right levels of abstraction for design primitives)

Change the frequency of activities (e.g. test less/test more)

Different input modalities (e.g. prompt engineering)

Generate data to define and detect new concepts crisply

Change order of activities

Merge activities

Examples of SEI Research in AI-Augmented Software Engineering

Understanding technical debt

➔ Generate data to define and detect new concepts crisply

Large scale refactoring

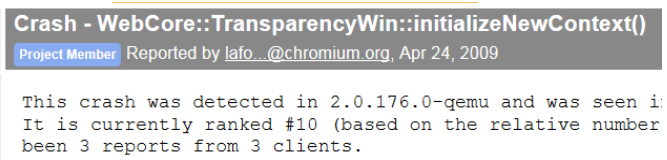
➔ Scope problem specific architecture evolution in large code bases

Architecture conformance

➔ Detect design pattern variations

Example SEI Research

Detecting Technical Debt
PI: Ipek Ozkaya



10977: Crash due to large negative number

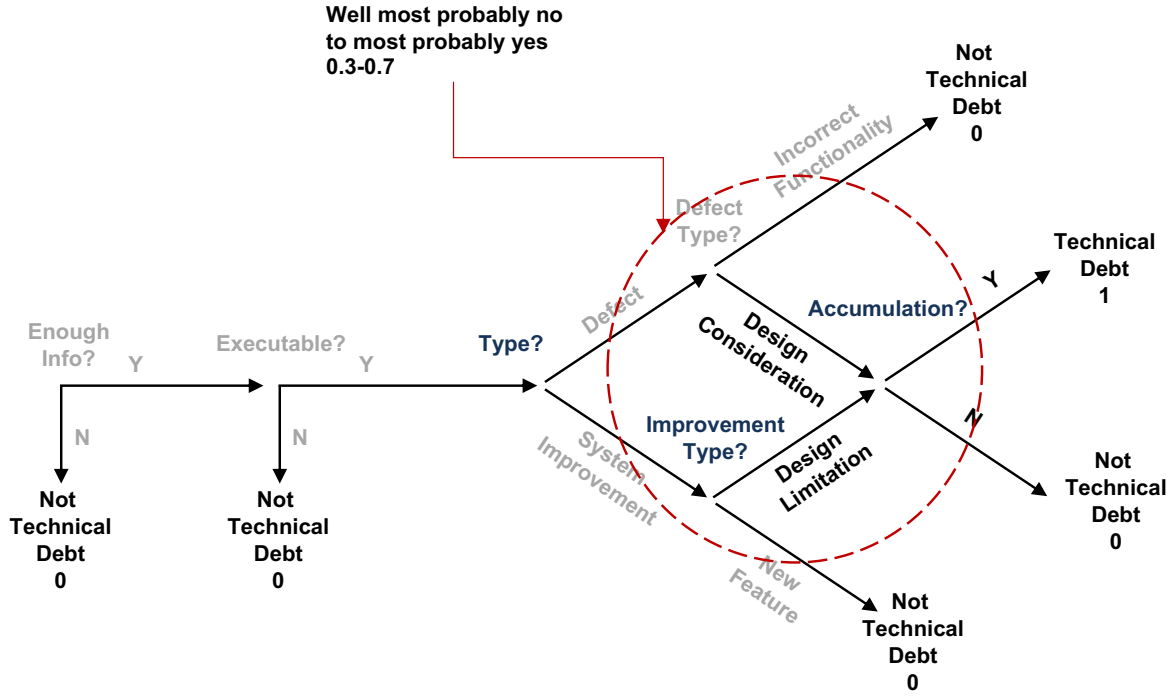
"*We could just fend off negative numbers near the crash site or we can dig deeper and find out how this -10000 is happening."*

"Time permitting, I'm inclined to want to know the root cause. My sense is that if we patch it here, it will pop up somewhere else later."

"There have been 28 reports from 7 clients… 18 reports from 6 clients."

"Hmm … reopening. The test case crashes a debug build, but not the production build. I have confirmed that the original source code does crash the production build, so there must be multiple things going on here."

Developing a Classifier



Detecting Discussions of Technical Debt
Ipek Ozkaya, Zachary Kurtz, Robert L. Nord, Raghvinder S. Sangwan, Satish M. Srinivasan
https://arxiv.org/abs/2201.12177

Boosting algorithms (LightGBM) to build the weighted average of many classification trees – iteratively improving weak classifiers and creating a final strong classifier

Active learning pipeline and iterating over the data set to use 1,934 labeled technical debt examples

Feature engineering to combine discussion length, *n*-grams, key phrases, concepts, and document context

Feature engineering

Ticket meta-data: status (duplicate, verified, fixed, wont fix…), authorship (to focus on developers on project), priority, type

Counts: length of comments

Key phrases: “debt,” “hack,” “workaround,” “cleanup,” “clean-up,” “clean up,” “give up,” “problematic,” “not up to date,” “inconsisten” [sic], “short term,” “deviate,” “tweak,” “mess,” “buggy,” “complex,” “doesn’t work,” “out of date,” “insufficient,” “rework,” “remove,” “redesign,” “refactor,” “depend,” and “structure.”

N-grams for n= 1, 2, 3.

Concept words: “deviate,” “outdated,” “redundant,” “redesign,” “decouple,” “complicated,” “regret,” “corrupt,” “horrible,” and “delay.”

Word and document vectors using gensim implementation of word2vec and doc2vec

Performance metrics

Using Chromium project with 475,000 issues

	Accuracy*	Precision*	Recall*	AUROC*
no TD	0.90	NA	0.00	0.50
keyphrase query	0.83	0.26	0.35	0.62
main model	0.87	0.40	0.62	0.88

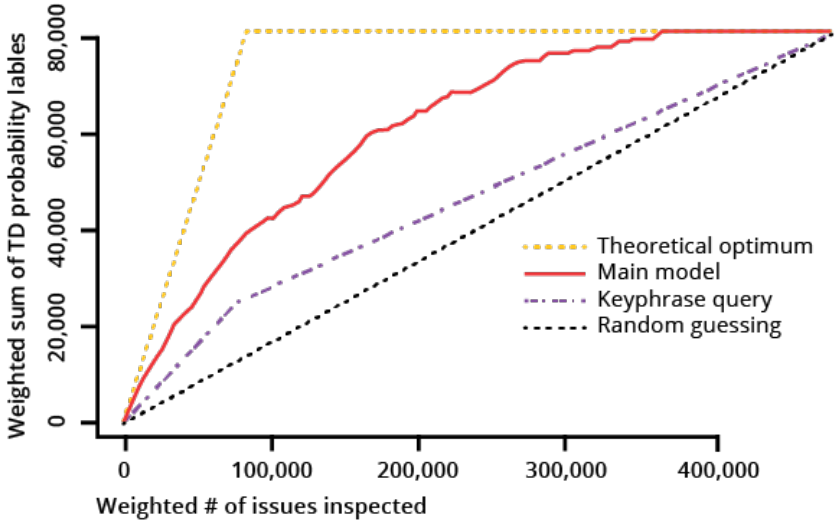
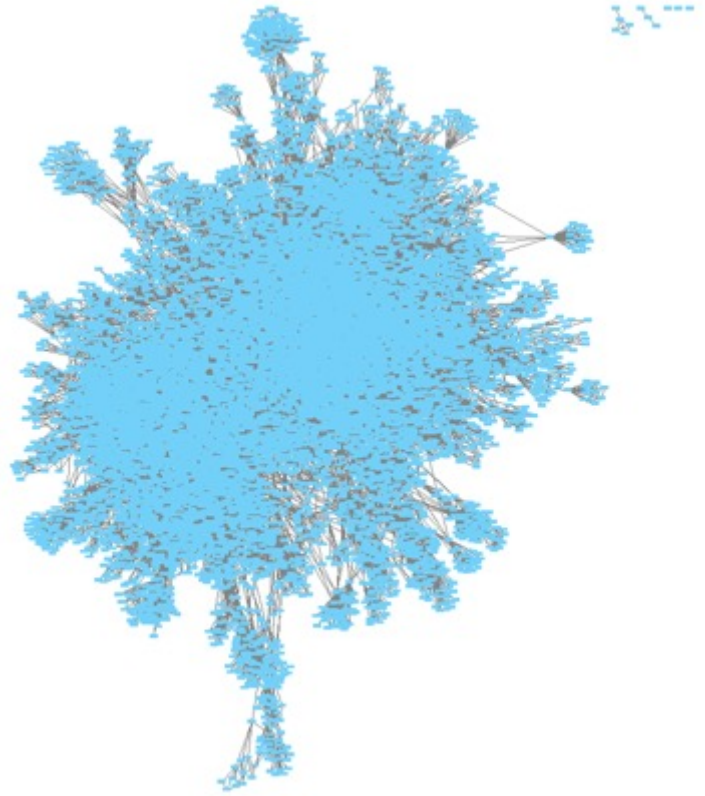


TABLE I
FEATURE TYPES RANKED BY IMPORTANCE

Rank	Feature Type	Number of Features	Total Gain
1	word vectors	19	0.407
2	key phrases	9	0.262
3	counts	6	0.184
4	concept words	10	0.072
5	document vectors	15	0.070
6	priority	1	0.003
7	author	2	0.002

Example SEI Research

Untangling the Knot
PI: James Ivers



An Automated Refactoring Assistant

We have developed an automated refactoring assistant for developers that improves software structure for several common forms of change that involve software isolation:

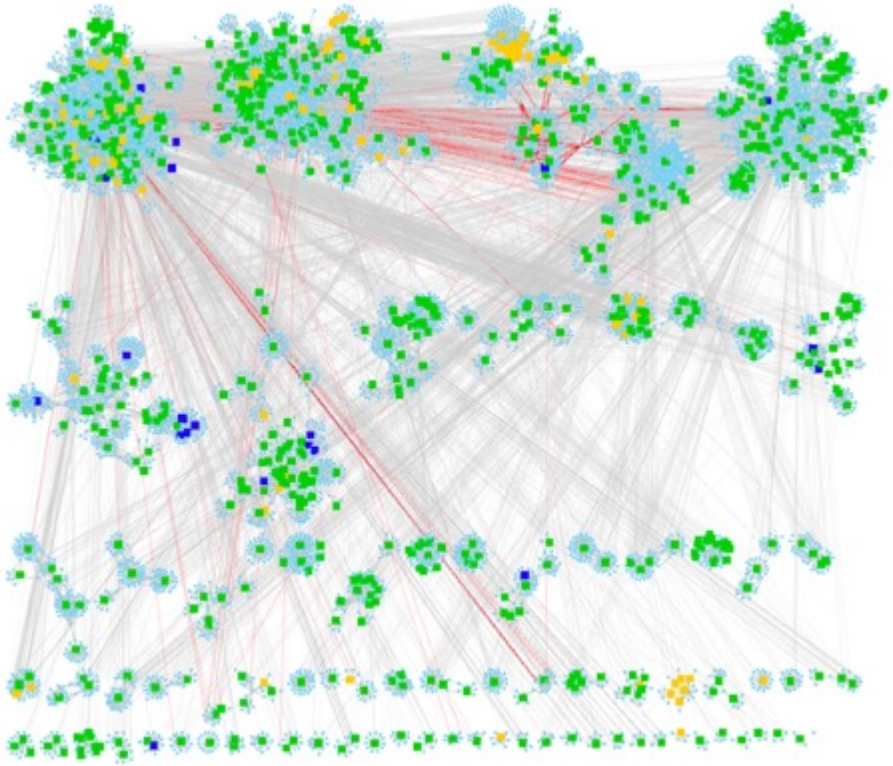
- Solves project-specific problems
- Uses a semi-automated approach
- Allows refactoring to be completed in less than 1/3 of the time required by manual approaches

Refactoring is a technique for improving the structure of software, but it is typically a *labor-intensive* process in which developers must

- figure out where changes are needed
- figure out which refactoring(s) to use
- implement refactorings by rewriting code

J. Ivers, I. Ozkaya, R. L. Nord, C. Seifried. **Next Generation Automated Software Evolution: Refactoring at Scale**. 2020. *28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20).*

Key Concept – Problematic Couplings



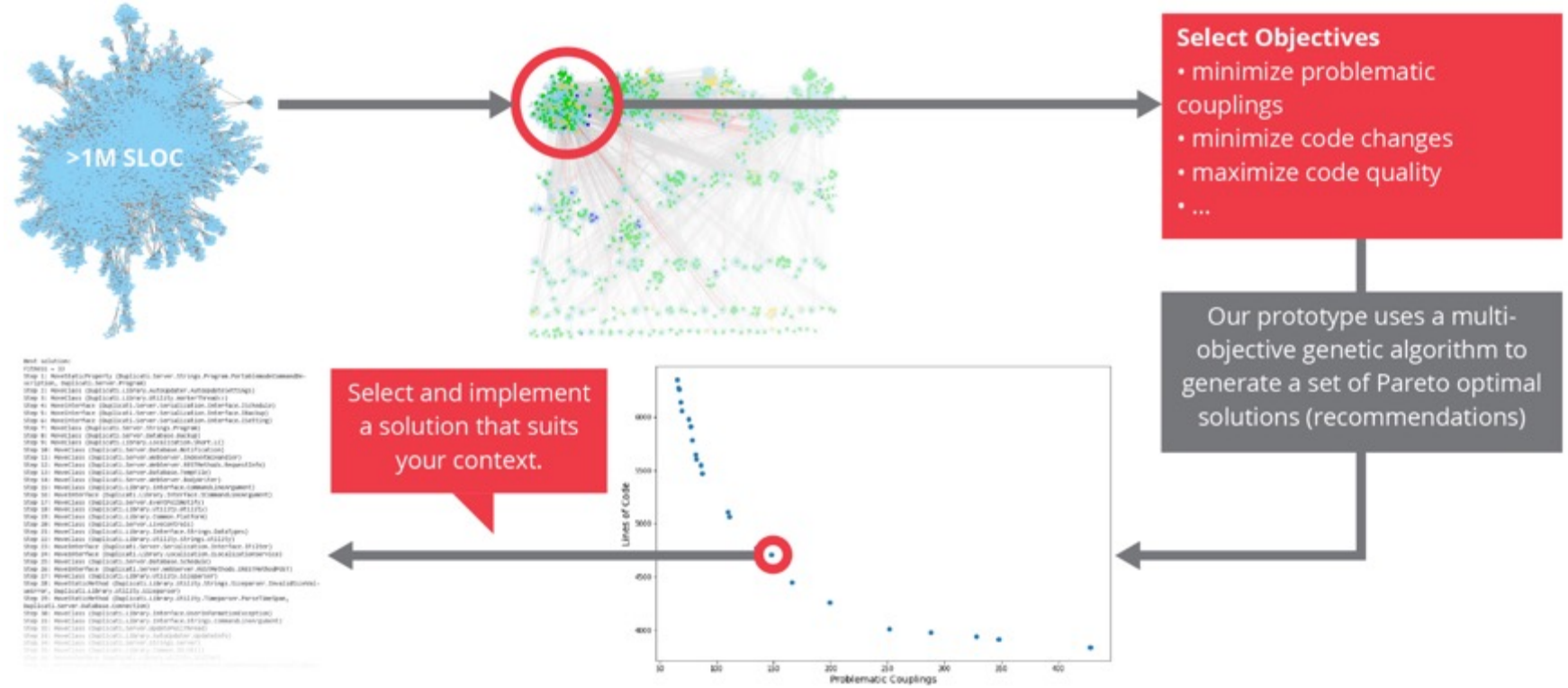
Only certain software dependencies interfere with any particular goal.

For example, if we want to harvest a feature:

- The core problem is dependencies (red lines) from software being harvested to software that is being left behind
- All other dependencies are irrelevant to the goal, allowing us to focus our analysis and search for solutions

This insight enables us to apply **search-based software engineering** techniques and treat this as an **optimization problem**.

Generating Refactoring Recommendations

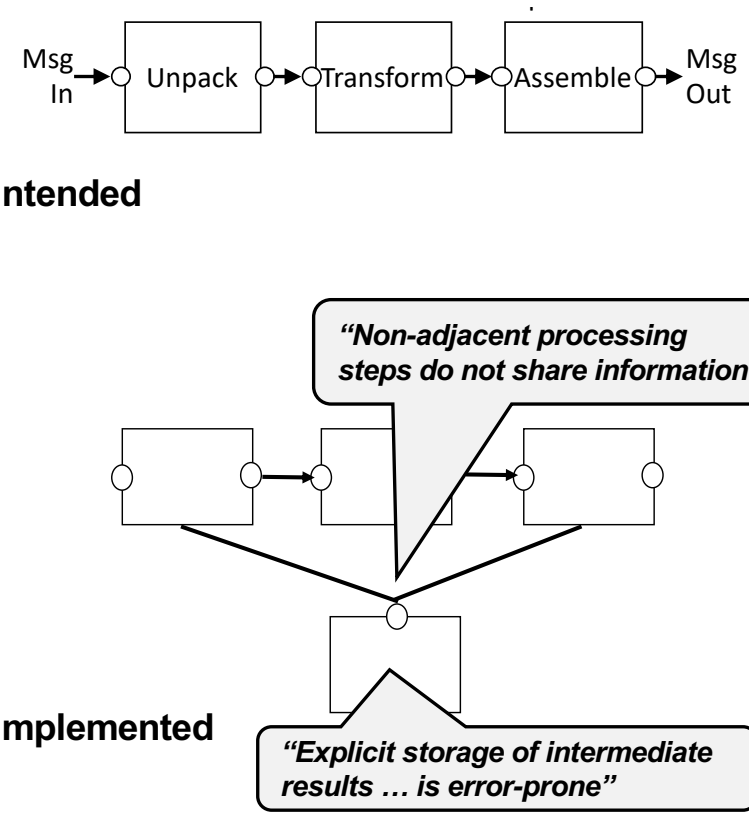


Untangling the knot: Enabling architecture evolution with search-based refactoring
J Ivers, C Seifried, I Ozkaya 2022 IEEE 19th International Conference on Software Architecture (ICSA), 101-111

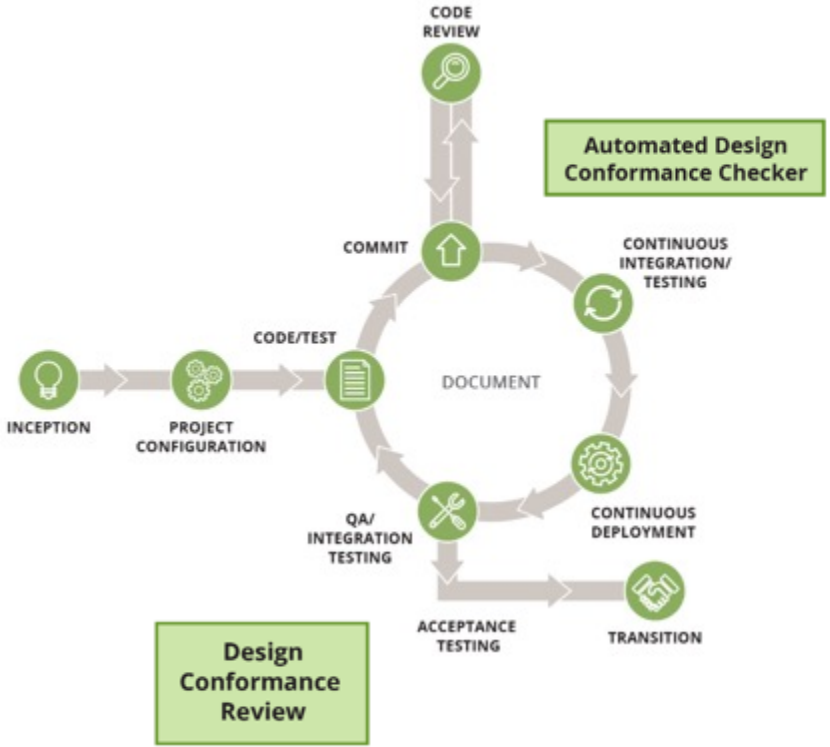
Example SEI Research

Automated Design Conformance

PI: Robert Nord



Automated Design Conformance during CI

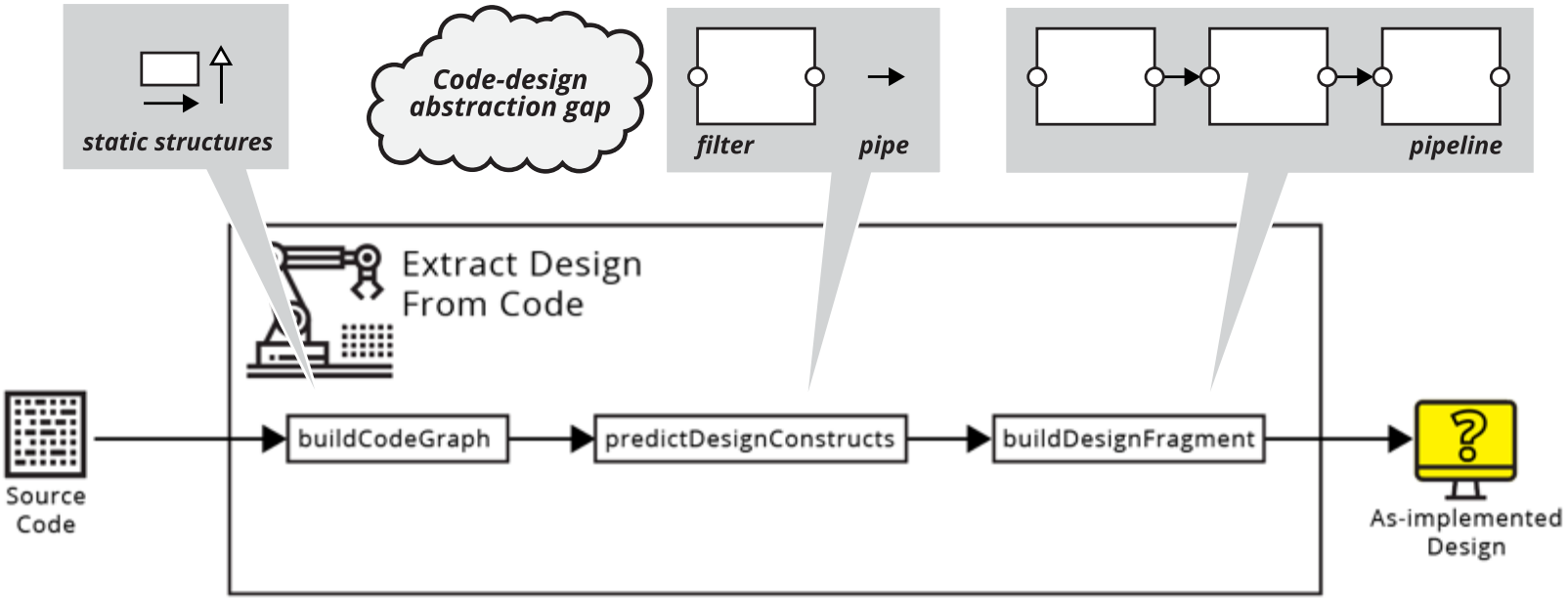


An automated design conformance checker integrated into a continuous integration workflow will reduce time to detect violations from months or years to hours.

Automation enables early detection and allows remediation before the violation gets “baked in” to the implementation.

Detection of nonconformances allows program managers to hold developers (contractor or organic) accountable.

Code-Design Abstraction Gap



Ivers J., Ozkaya, I, Nord, R. . (2019). **Can AI Close the Design-Code Abstraction Gap?** *International Workshop on Software Engineering Intelligence, IEEE/ACM International Conference on Automated Software Engineering (ASE).*

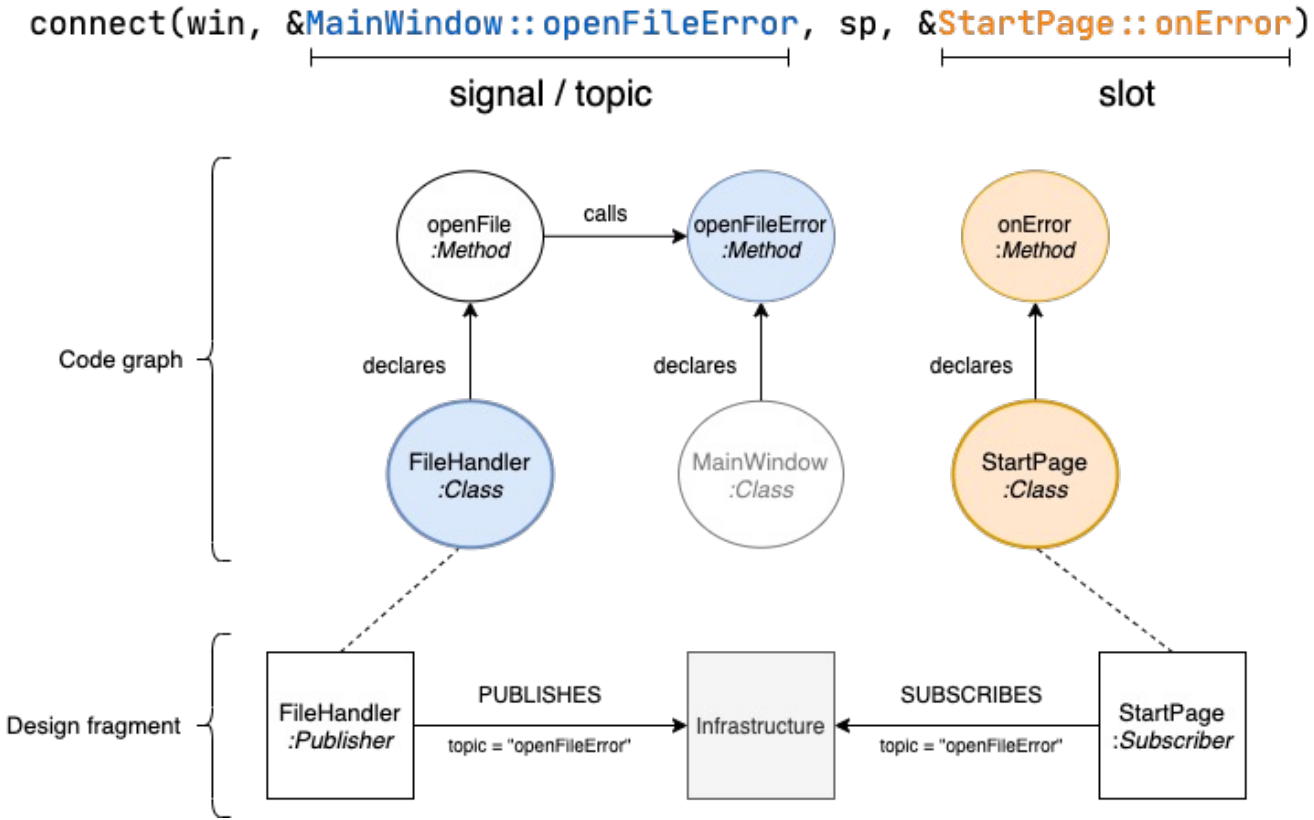
From Code to Design Fragment

How do you recognize design abstractions from code?

- Rules or classifiers?
- Based on what data?
- How generalizable can you get?

Hotspot (Qt)
github.com/KDAB/hotspot

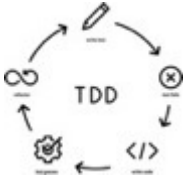
- **8K** code lines
- **2,648** nodes and **11,427** relations
- **7** publishers, **37** subscribers



How should an AI-augment Software Development Lifecycle be structured?

Key insight: do all tasks in smaller scope of requirements, iterate and incrementally grow as opposed to tackling entire scope conducting all activities consecutively

Key insight: write tests first, implement against them, run all tests as you develop as opposed to writing and running tests at the end of implementation



Key insight: integrate continuously as opposed to at the end of the development, run all checks during each small integration as opposed to during predetermined phases

Data is the loudest person in the room!

Quote credit: Lena Pons, ML researcher CMU SEI

What are the right units of abstraction?

Does scope of work, batch size change (line of code, class, story, pull request, design construct….)?

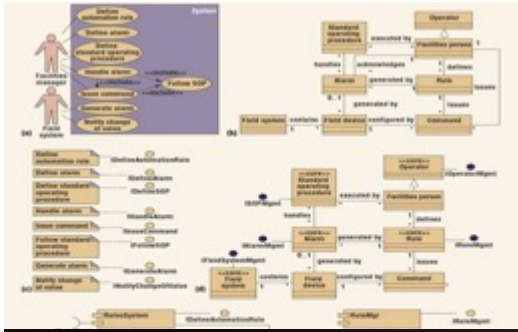
Where does the ground truth get created (your code, your architecture, other peoples code, specifications)

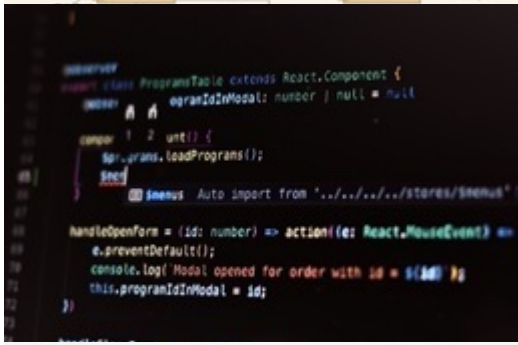
What do you look for (correctness versus mistakes)?

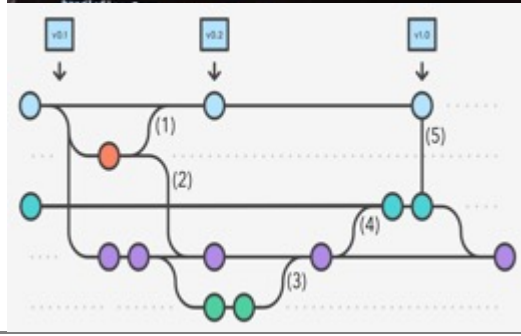
What data exist or need to be collected?

How do different roles interact with work and collaborate with each other?

What is a good unit of design?







Observations

There is no one size fits all!

- A collection of generative AI, other AI and approaches will need to be orchestrated together.

Start with the problem not the solution!

- Not all software engineering problems are fit for ML or generative AI

Decompose complex tasks into tedious tasks!

- A tedious task is repetitive, numerous, and have bounded decision space
- Tedious tasks are great for automation and AI support

A Paradigm Shift in Automating Software Engineering Tasks: Bots
I Ozkaya IEEE Software 39 (5), 4-8

AI-Augmented Software Development
Open Research Challenges

Prompt engineering

Training domain specific LLMs

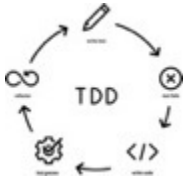
Human-computer partnership, trust, and workflows

Automatically accumulating and carrying along evidence of quality; verifying results are correct at different steps in the process

Obtaining data to model for different software engineering stage and workflows

IDEs that incorporate AI-assistants, multi-modal interactions

How should an AI-augment Software Development Lifecycle be structured?

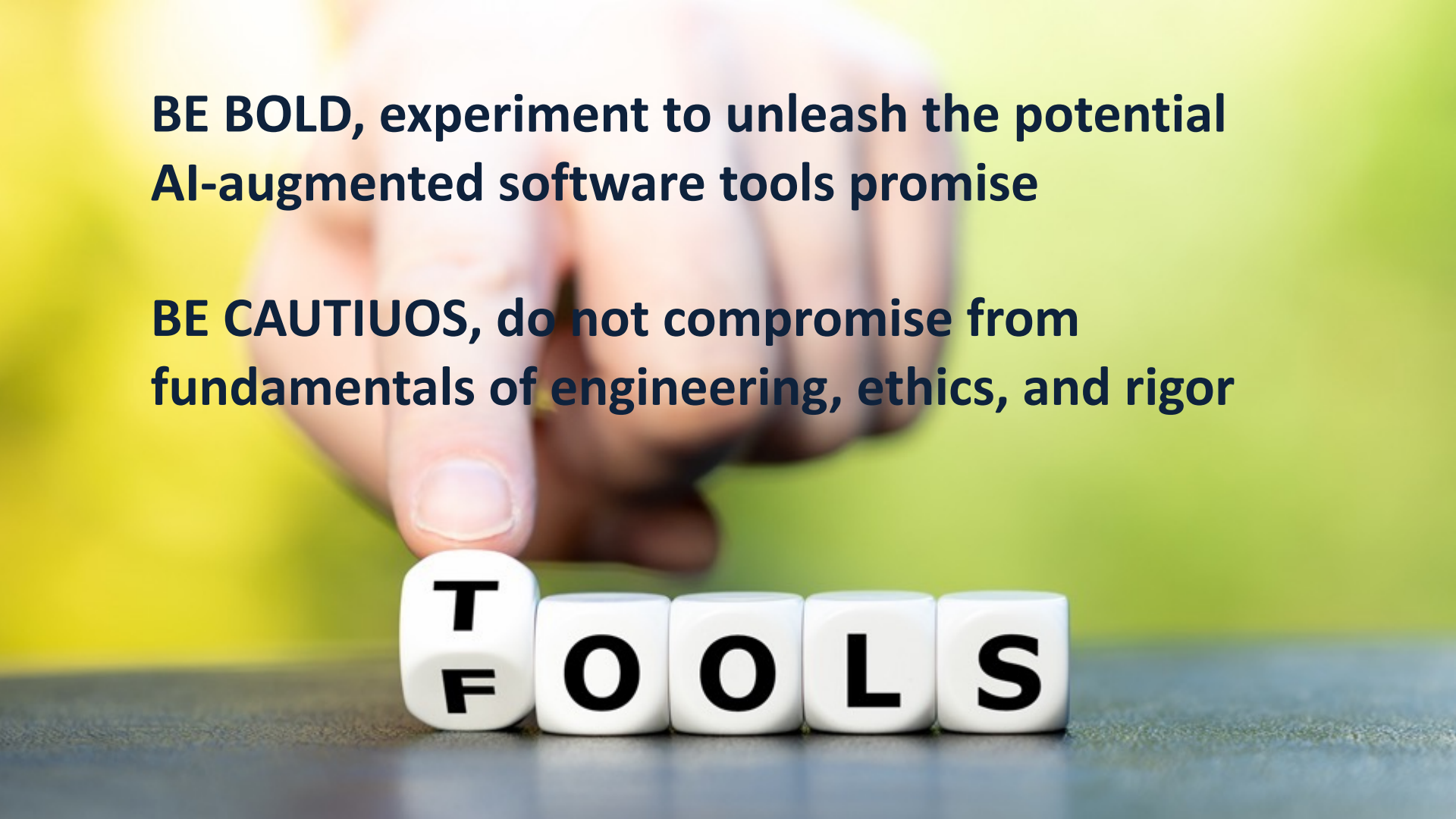


Challenges: Speed and correctness at scale, respond to change at scale, trust, design competence

Key insight: ?

BE BOLD, experiment to unleash the potential AI-augmented software tools promise

BE CAUTIUOS, do not compromise from fundamentals of engineering, ethics, and rigor



THANK YOU!