CodeScene

Refactoring vs Refuctoring

Code quality in the AI age

Enys Mones & Peter Anderberg

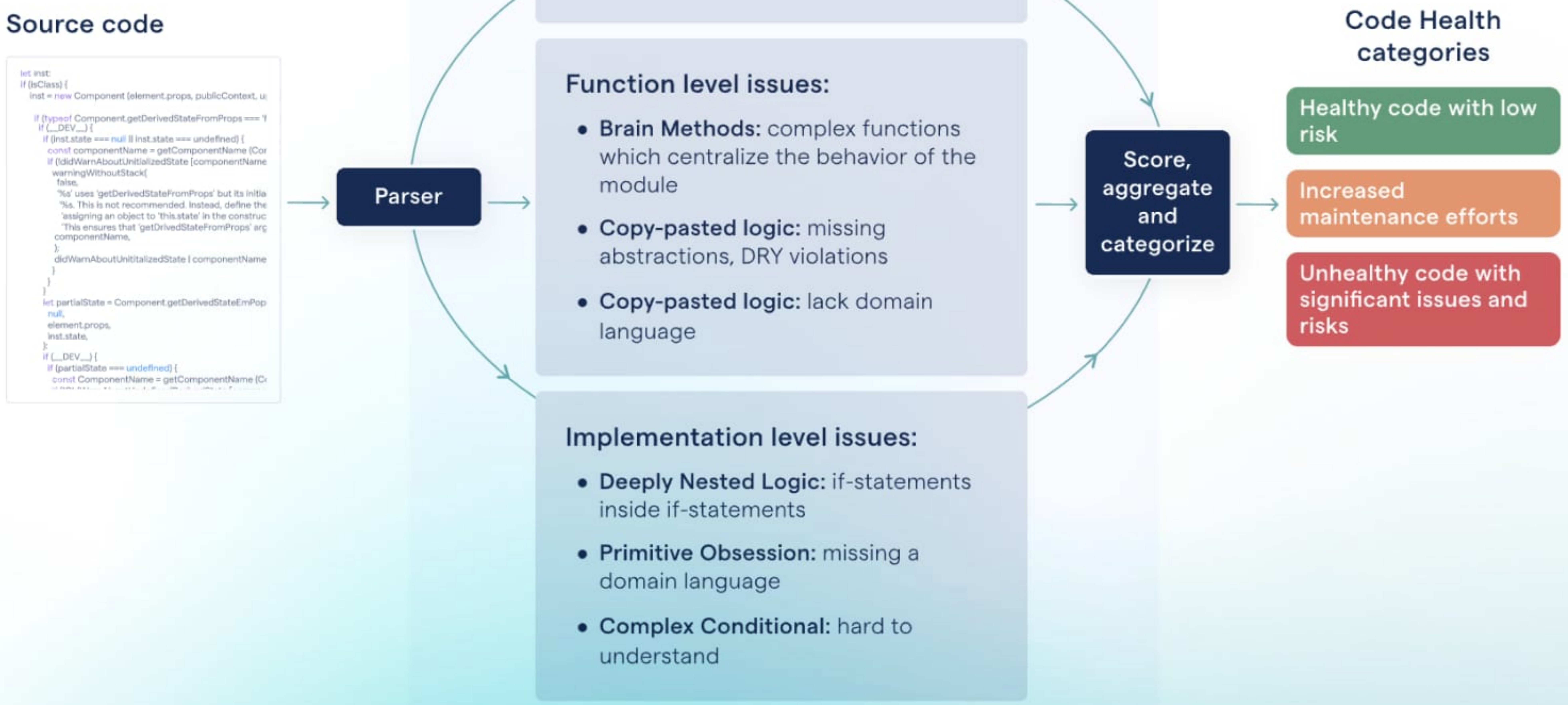
codescene.com

Over the coming decades, we’ll have a hybrid of code written by both humans and machines. Who has the overall mental model in that context, and how do we ensure our AI generates human-readable code? To face the challenge, we need a safety net to enforce healthy code.

Adam Tornhill

Quality matters

Code Health



Source code

```
let inst:
if (isClass) {
  inst = new Component (element.props, publicContext, u
  
   if (typeof Component.getDerivedStateFromProps === 'f
    if (__DEV__) {
     if (inst.state === null || inst.state === undefined) {
      const componentName = getComponentName (Cor
      if (!didWarnAboutUnitializedState [componentName
       warningWithoutStack(
        false,
        '%s' uses 'getDerivedStateFromProps' but its initia
         '%s. This is not recommended. Instead, define the
          'assigning an object to 'this.state' in the construc
          'This ensures that 'getDrivedStateFromProps' arg
        componentName,
        );
        didWarnAboutUnititalizedState | componentName
       }
      }
     }
    let partialState = Component.getDerivedStateEmPop
      null,
      element.props,
      inst.state,
    );
    if (__DEV__) {
      if (partialState === undefined) {
       const ComponentName = getComponentName (C
```

Parser

Examples on unhealthy code

Module level issues:

- **Low Cohesion:** many responsibilities
- **Brain Class:** low cohesion + large class + at least one Brain Method
- **Lack of Modularity:** too many business aspects

Function level issues:

- **Brain Methods:** complex functions which centralize the behavior of the module
- **Copy-pasted logic:** missing abstractions, DRY violations
- **Copy-pasted logic:** lack domain language

Implementation level issues:

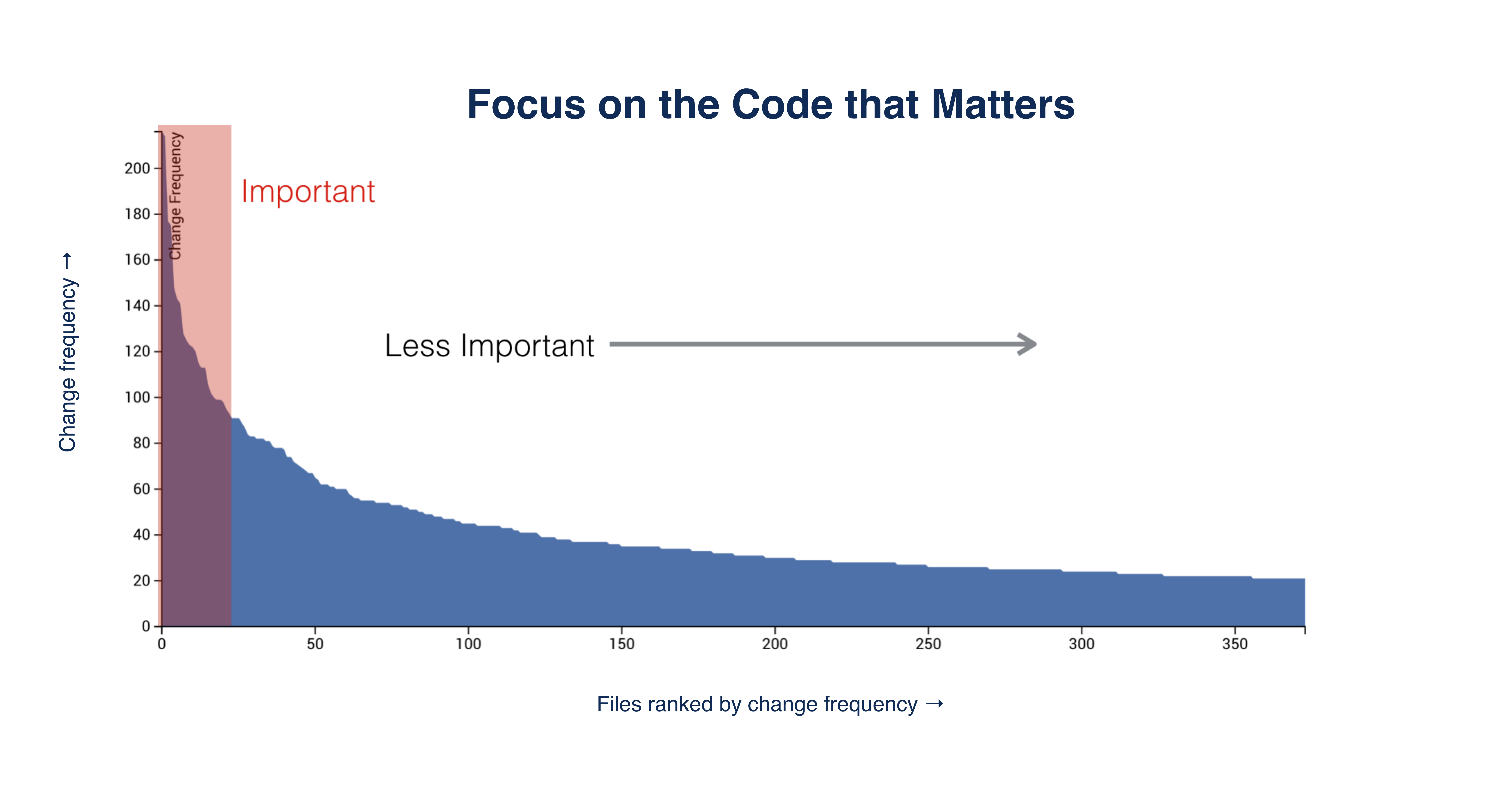
- **Deeply Nested Logic:** if-statements inside if-statements
- **Primitive Obsession:** missing a domain language
- **Complex Conditional:** hard to understand

Score, aggregate and categorize

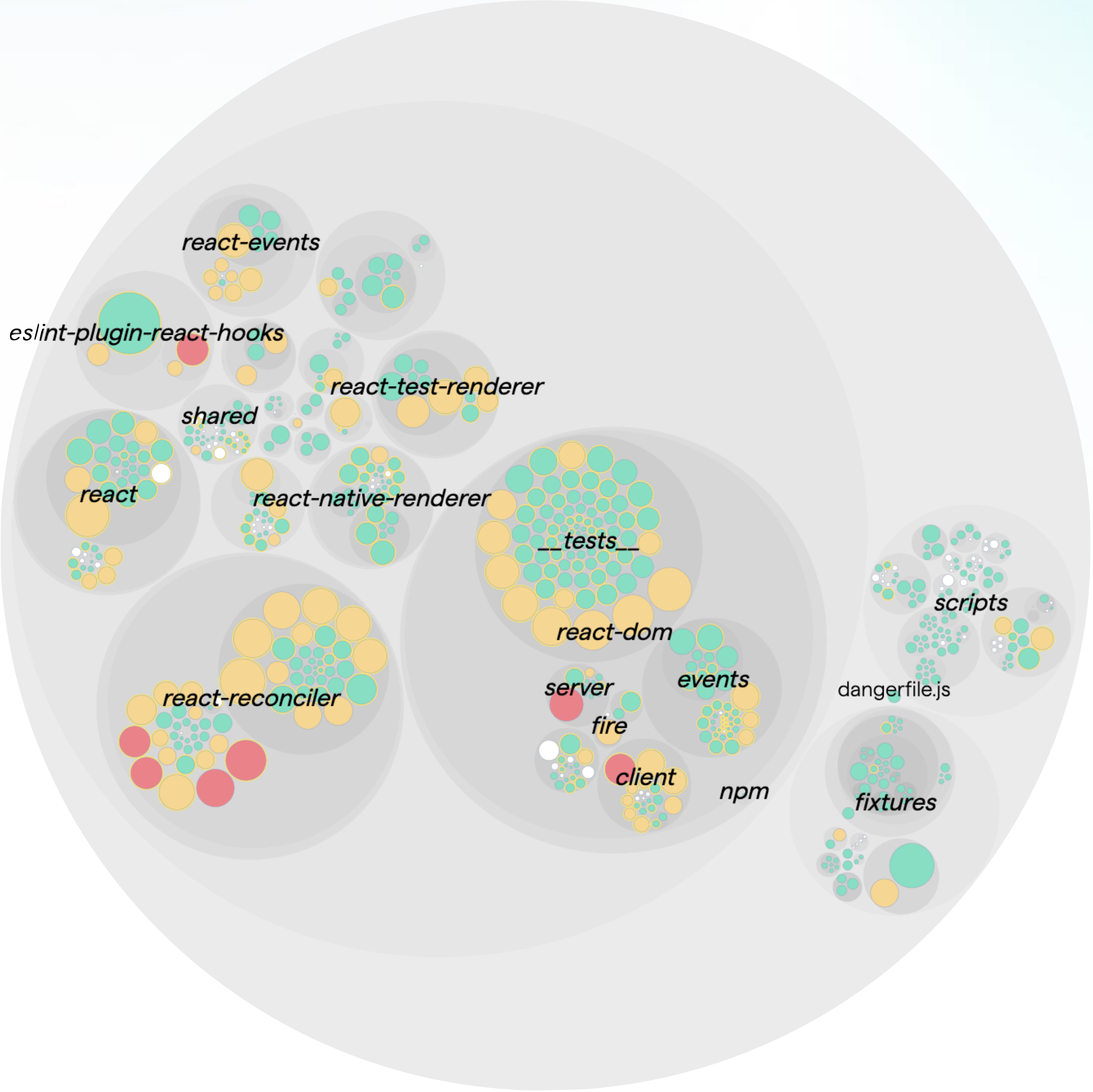
Code Health categories

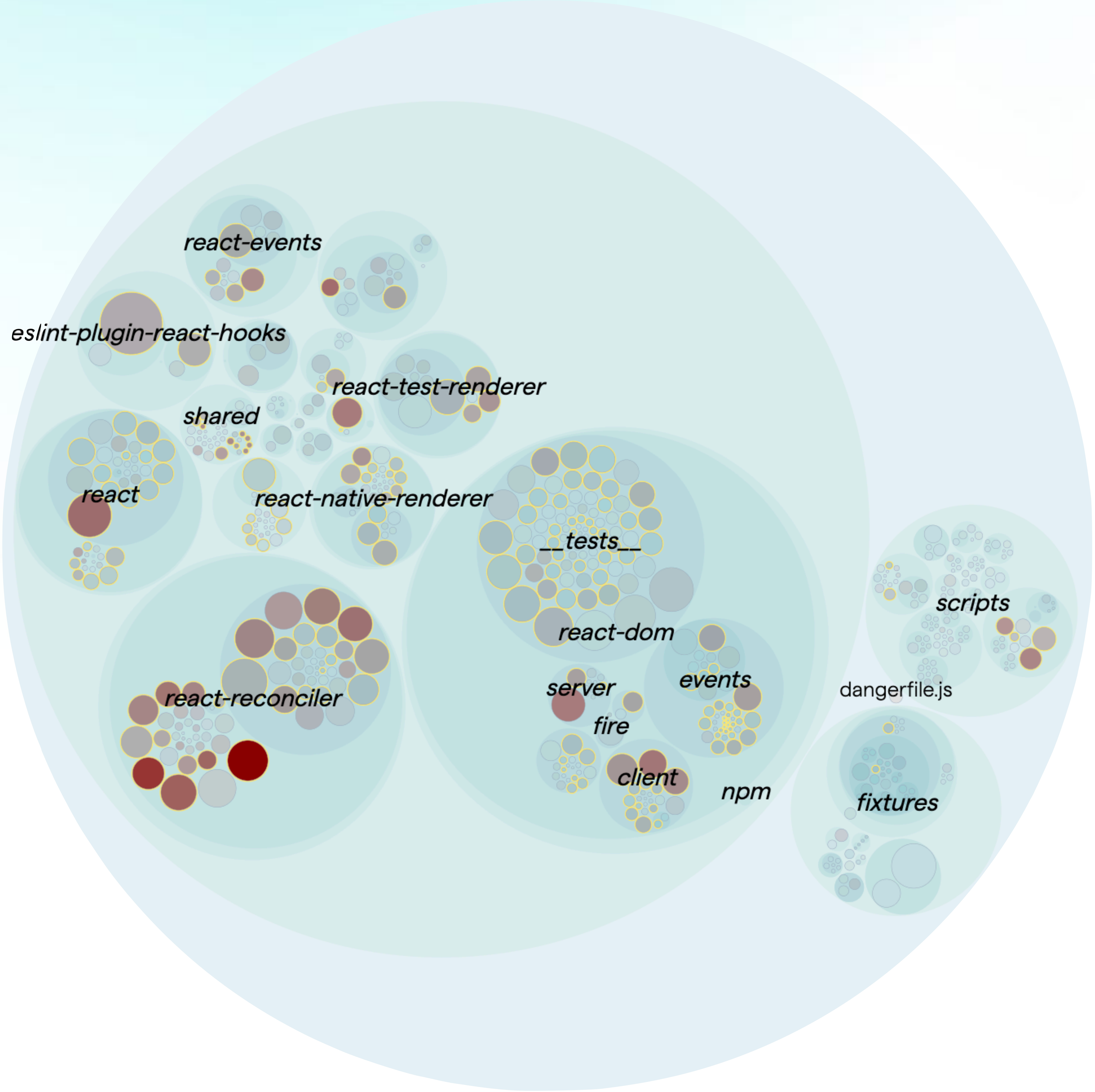
- Healthy code with low risk
- Increased maintenance efforts
- Unhealthy code with significant issues and risks

Important code



Code Health - Relevance





ⓘ ● Healthy ● Problematic ● Unhealthy ...

ⓘ Low development activity — Hotspot

Why all of these matter after all? - Code Red

Whitepaper

Code Red:
The business impact of low code quality

This paper presents data from a large-scale study on how code quality impacts software companies in terms of time-to-market and product experience. We conclude with an analysis of the impact and specific recommendations towards successful software development.

Target audience

- Business managers
- Product owners/managers
- Technical managers
- Tech leads
- Development teams

About CodeScene

CodeScene is the intersection of code and people, empowering companies to build great software.

CodeScene was born in 2015 when founder Adam Tornhill published the book "Your Code as a Crime Scene". It introduced a new approach to software analysis which focused on the evolution of a codebase over time.

CodeScene has become the next generation of code analysis and is used by global Fortune 100 companies in a wide variety of domains.

CodeScene

Quantitative study of code quality impact

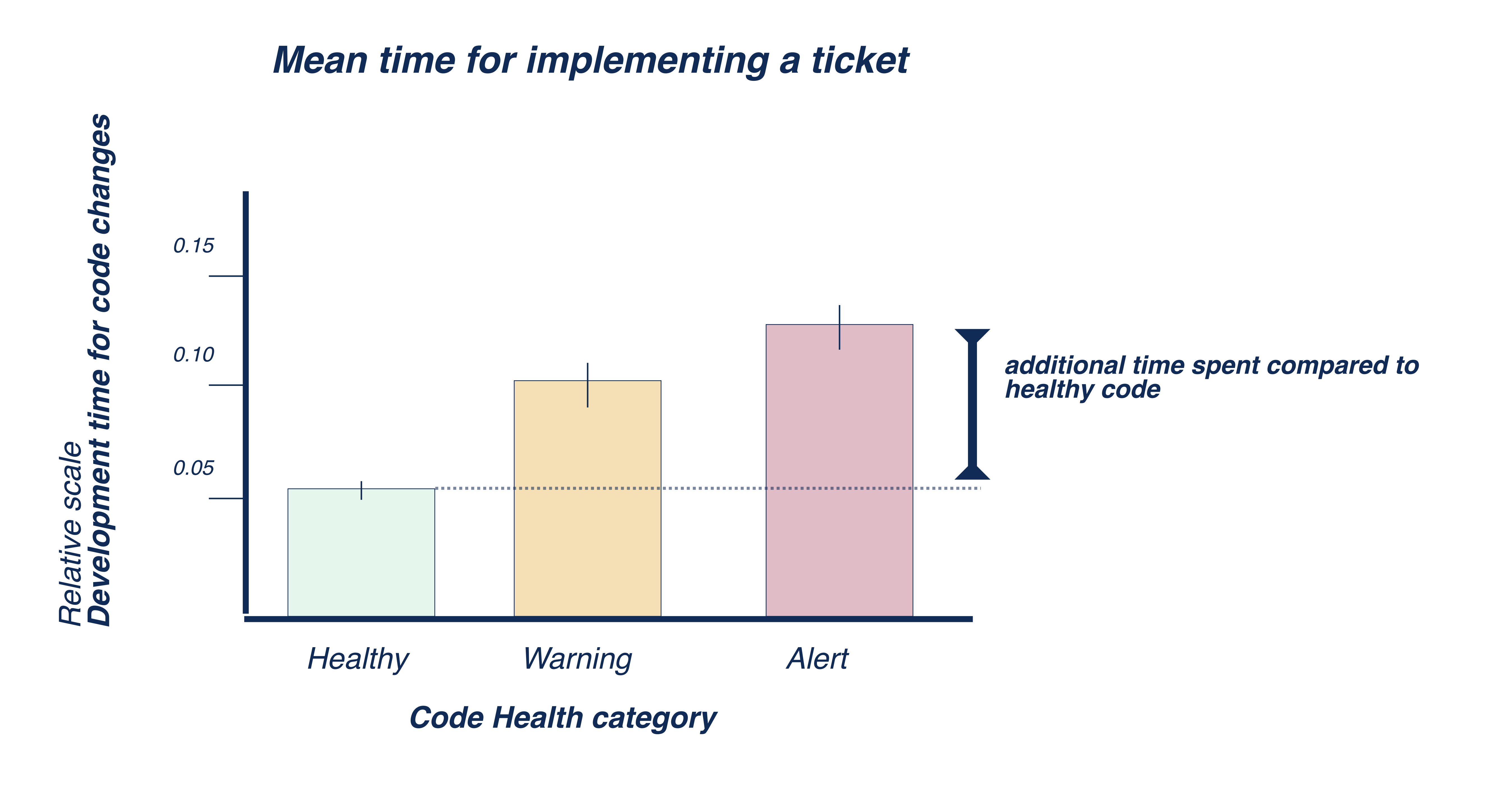
Many different industry segments

39 commercial codebases

40k+ software modules

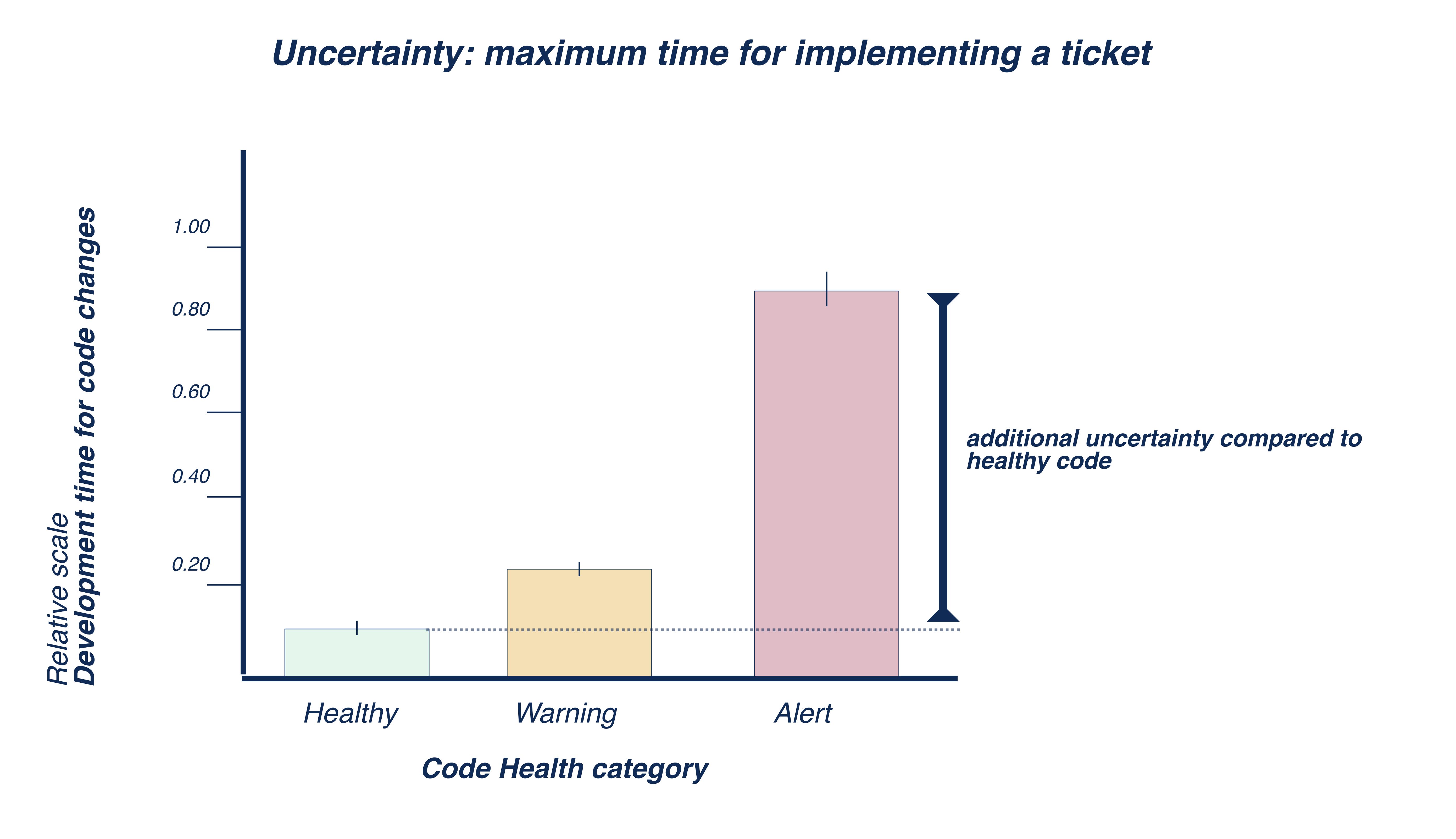
14 programming languages

Why all of these matter after all? - Code Red



Tornhill, A. & Borg, M. (2022) Code Red: The Business Impact of Code Quality
https://arxiv.org/abs/2203.04374

Why all of these matter after all? - Code Red



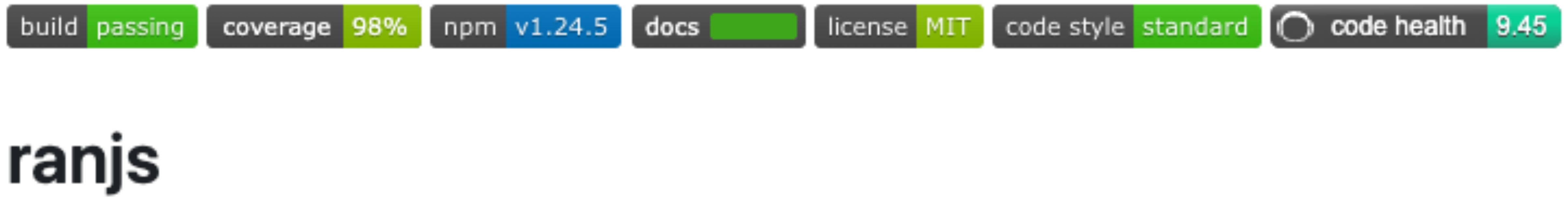
Tornhill, A. & Borg, M. (2022) Code Red: The Business Impact of Code Quality
https://arxiv.org/abs/2203.04374

Better code quality leads to faster development

Hybrid coding

The new era

Few years back, I started a new hobby...



ranjs

Statistical methods

Mathematically involved functions

Statistically sound tests and high precision

The new era

Few years back, I started a new hobby...

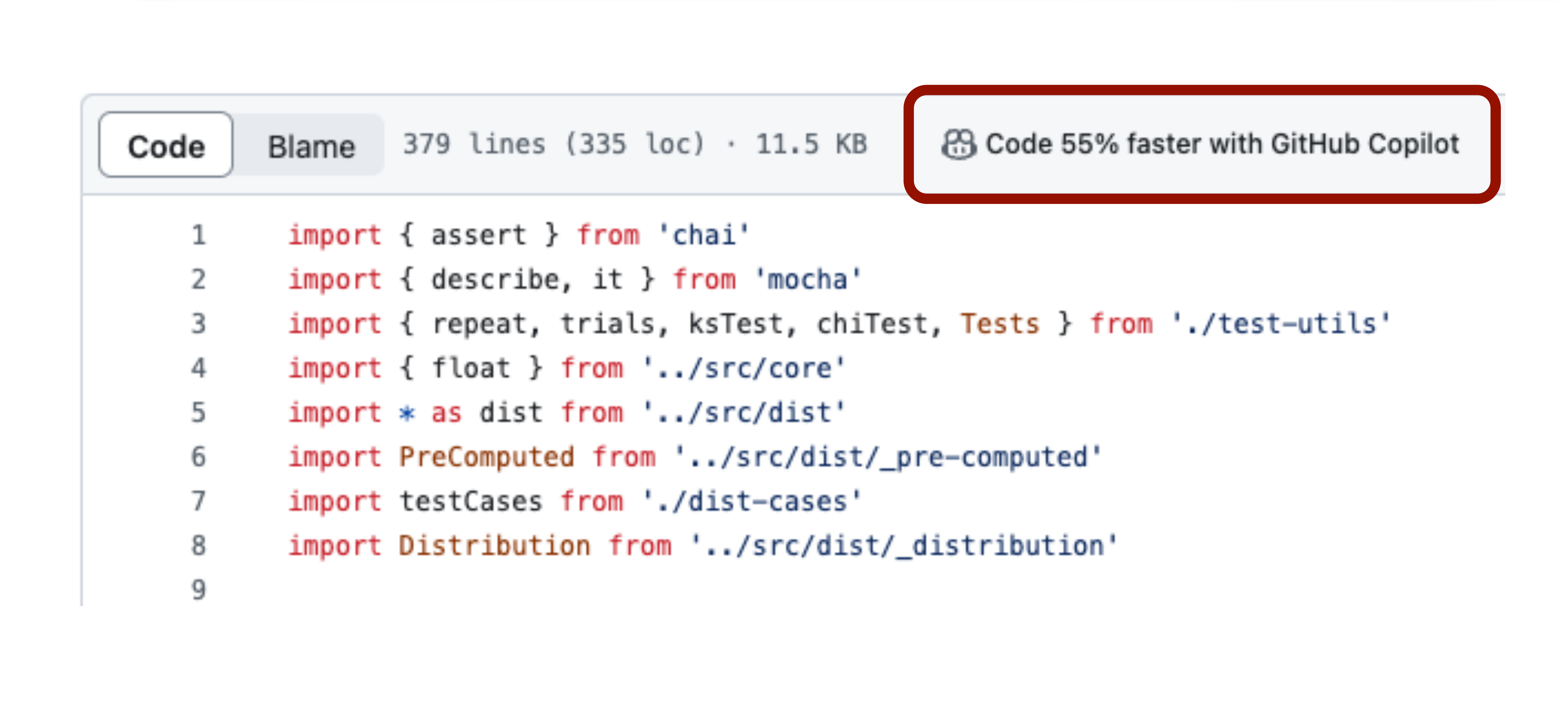
build passing coverage 98% npm v1.24.5 docs license MIT code style standard code health 9.45

ranjs

Statistical methods

Mathematically involved functions

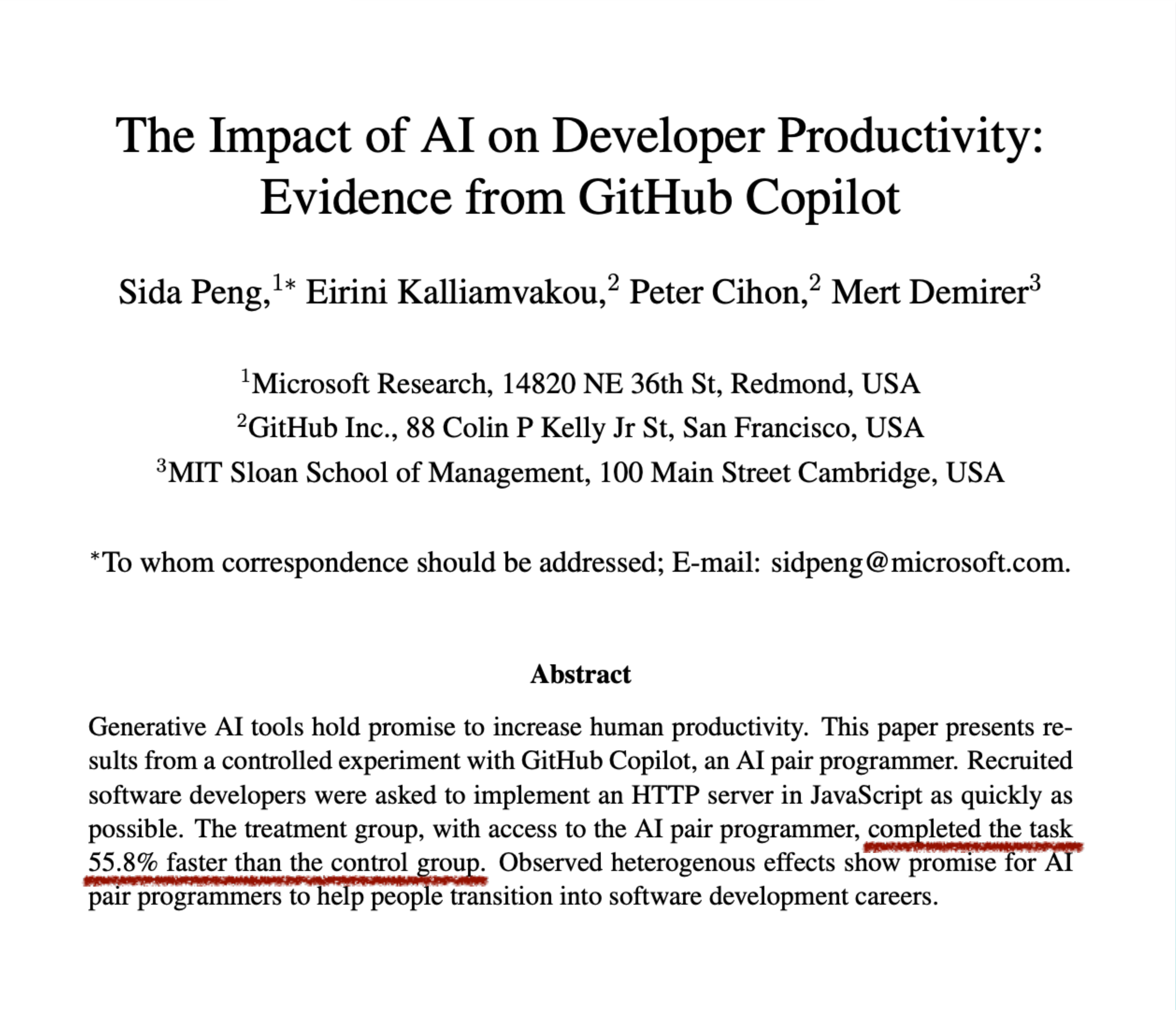
Statistically sound tests and high precision


Code
Blame
379 lines (335 loc) · 11.5 KB
Code 55% faster with GitHub Copilot

```
import { assert } from 'chai'
import { describe, it } from 'mocha'
import { repeat, trials, ksTest, chiTest, Tests } from './test-utils'
import { float } from '../src/core'
import * as dist from '../src/dist'
import PreComputed from '../src/dist/_pre-computed'
import testCases from './dist-cases'
import Distribution from '../src/dist/_distribution'

```

AI assisted coding



The Impact of AI on Developer Productivity: Evidence from GitHub Copilot

Sida Peng,[1]* Eirini Kalliamvakou,[2] Peter Cihon,[2] Mert Demirer[3]

[1]Microsoft Research, 14820 NE 36th St, Redmond, USA
[2]GitHub Inc., 88 Colin P Kelly Jr St, San Francisco, USA
[3]MIT Sloan School of Management, 100 Main Street Cambridge, USA

*To whom correspondence should be addressed; E-mail: sidpeng@microsoft.com.

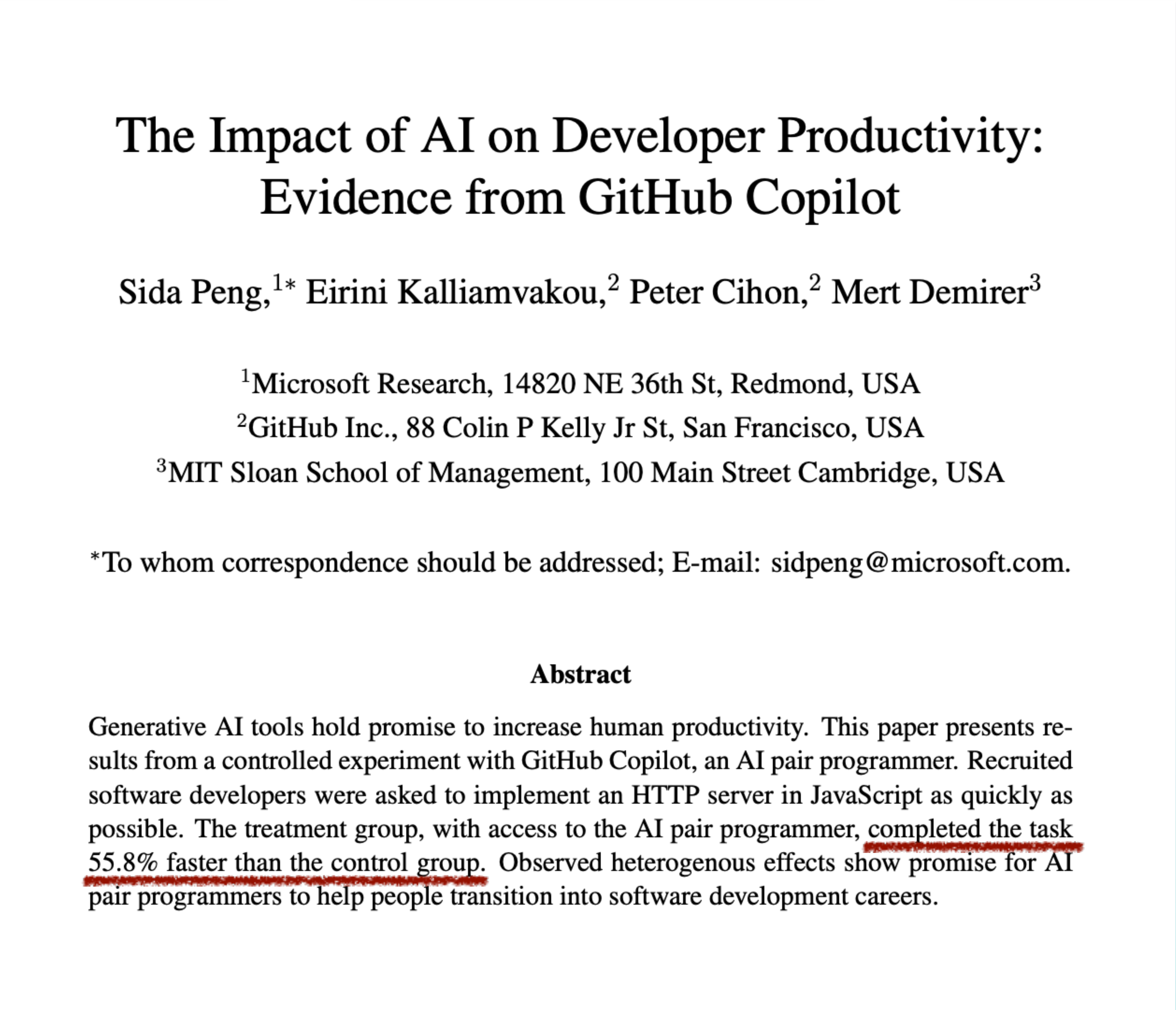
Abstract

Generative AI tools hold promise to increase human productivity. This paper presents results from a controlled experiment with GitHub Copilot, an AI pair programmer. Recruited software developers were asked to implement an HTTP server in JavaScript as quickly as possible. The treatment group, with access to the AI pair programmer, completed the task 55.8% faster than the control group. Observed heterogenous effects show promise for AI pair programmers to help people transition into software development careers.

"Productivity benefits may vary across specific tasks and programming languages, so **more research is needed to understand how our results generalizes** to other tasks."

https://arxiv.org/pdf/2302.06590.pdf

AI assisted coding



The Impact of AI on Developer Productivity: Evidence from GitHub Copilot

Sida Peng,[1]* Eirini Kalliamvakou,[2] Peter Cihon,[2] Mert Demirer[3]

[1]Microsoft Research, 14820 NE 36th St, Redmond, USA
[2]GitHub Inc., 88 Colin P Kelly Jr St, San Francisco, USA
[3]MIT Sloan School of Management, 100 Main Street Cambridge, USA

*To whom correspondence should be addressed; E-mail: sidpeng@microsoft.com.

Abstract

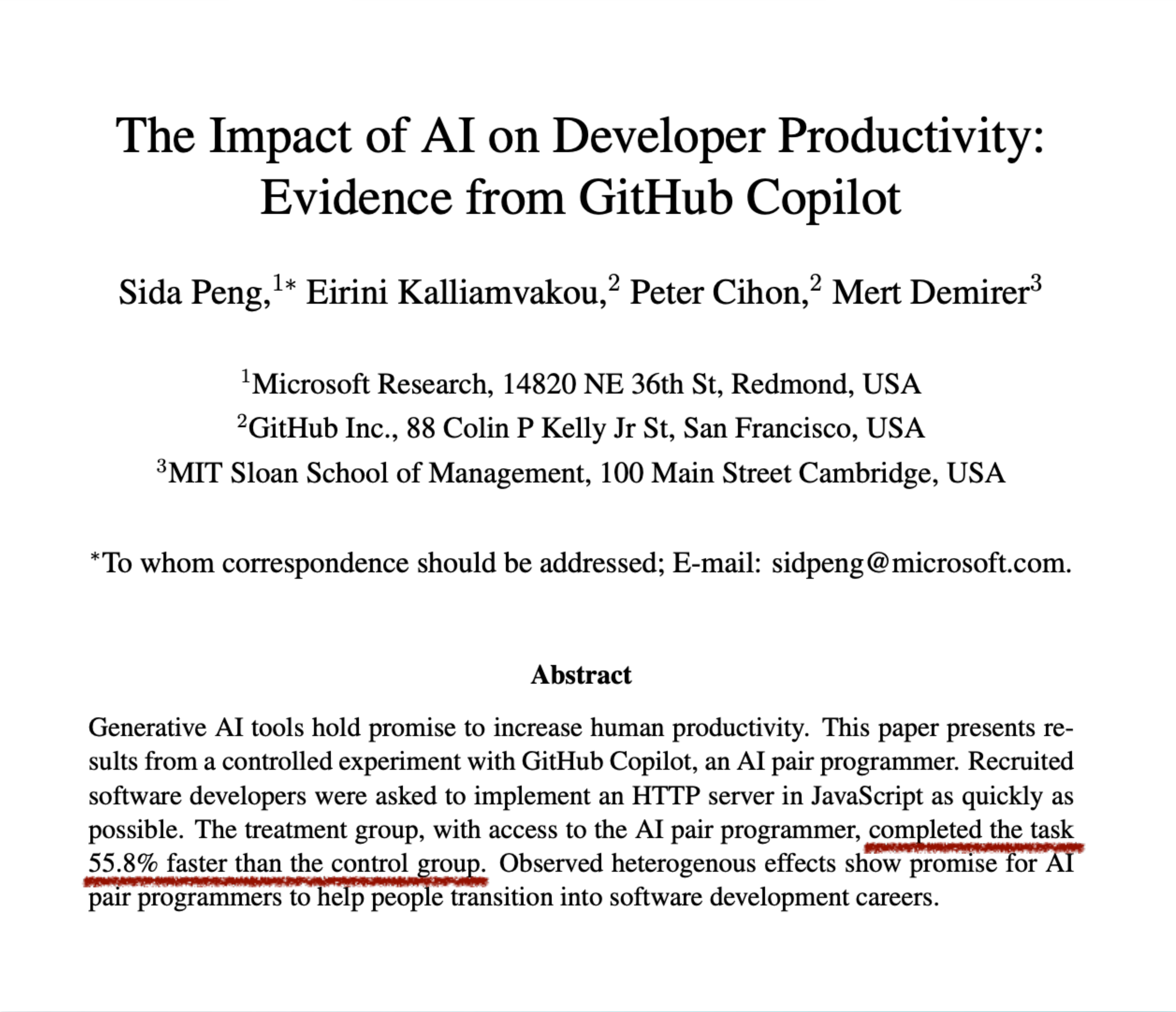
Generative AI tools hold promise to increase human productivity. This paper presents results from a controlled experiment with GitHub Copilot, an AI pair programmer. Recruited software developers were asked to implement an HTTP server in JavaScript as quickly as possible. The treatment group, with access to the AI pair programmer, completed the task 55.8% faster than the control group. Observed heterogenous effects show promise for AI pair programmers to help people transition into software development careers.

"Productivity benefits may vary across specific tasks and programming languages, so **more research is needed to understand how our results generalizes** to other tasks."

"Our results suggest that **less experienced programmers benefit more** from Copilot."

https://arxiv.org/pdf/2302.06590.pdf

AI assisted coding



The Impact of AI on Developer Productivity: Evidence from GitHub Copilot

Sida Peng,[1*] Eirini Kalliamvakou,[2] Peter Cihon,[2] Mert Demirer[3]

[1]Microsoft Research, 14820 NE 36th St, Redmond, USA
[2]GitHub Inc., 88 Colin P Kelly Jr St, San Francisco, USA
[3]MIT Sloan School of Management, 100 Main Street Cambridge, USA

*To whom correspondence should be addressed; E-mail: sidpeng@microsoft.com.

Abstract

Generative AI tools hold promise to increase human productivity. This paper presents results from a controlled experiment with GitHub Copilot, an AI pair programmer. Recruited software developers were asked to implement an HTTP server in JavaScript as quickly as possible. The treatment group, with access to the AI pair programmer, completed the task 55.8% faster than the control group. Observed heterogenous effects show promise for AI pair programmers to help people transition into software development careers.

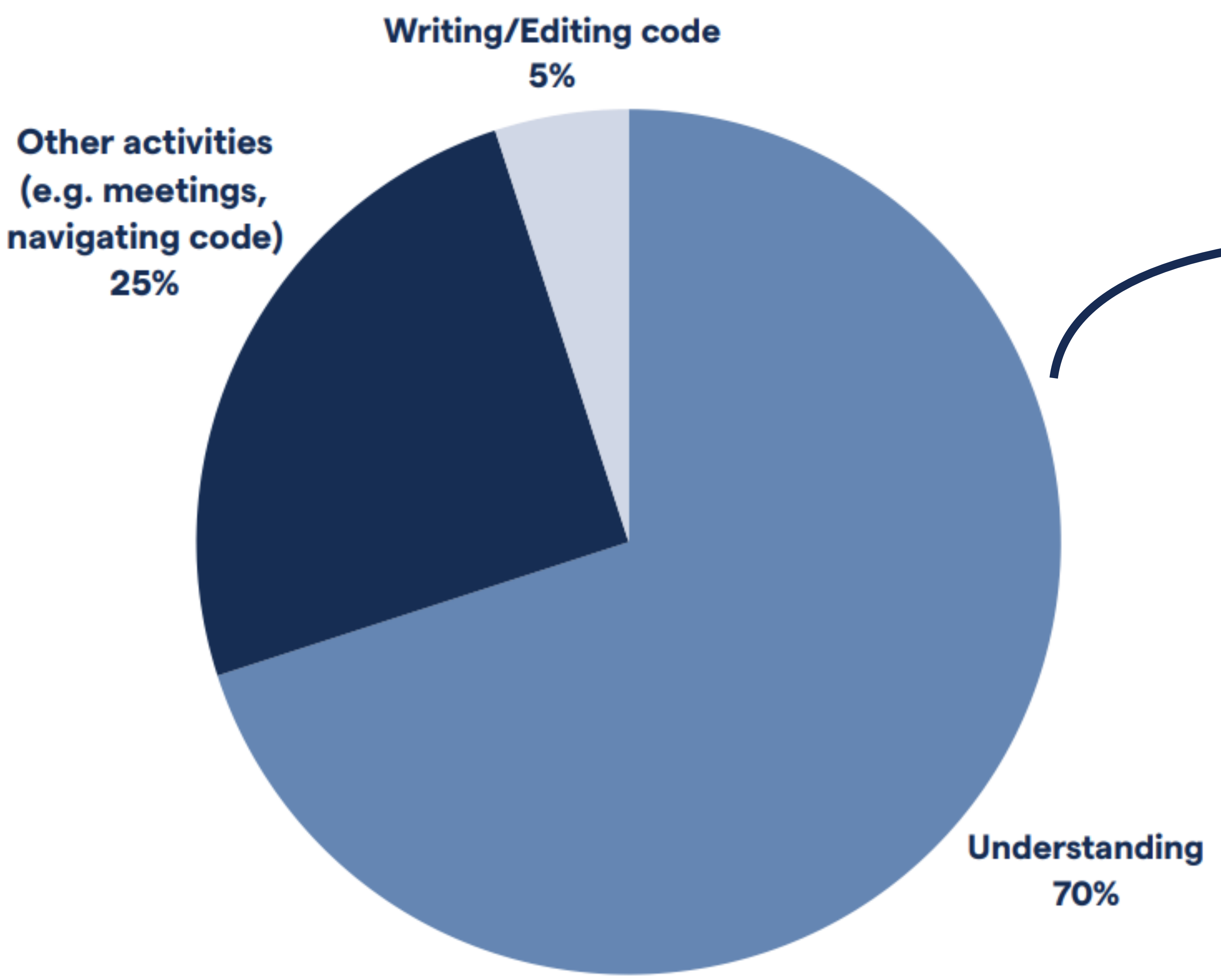
“Productivity benefits may vary across specific tasks and programming languages, so **more research is needed to understand how our results generalizes** to other tasks.”

“Our results suggest that **less experienced programmers benefit more** from Copilot.”

“Finally, this study does not examine the **effects of AI on code quality**.“

https://arxiv.org/pdf/2302.06590.pdf

The bigger picture of time spent



Today's focus: **new code**

Big win: **existing code**

55% faster on this part means ~1 hour saved per work week

The majority of a developer's time is spent trying to understand the existing system

Minelli et al., 2015

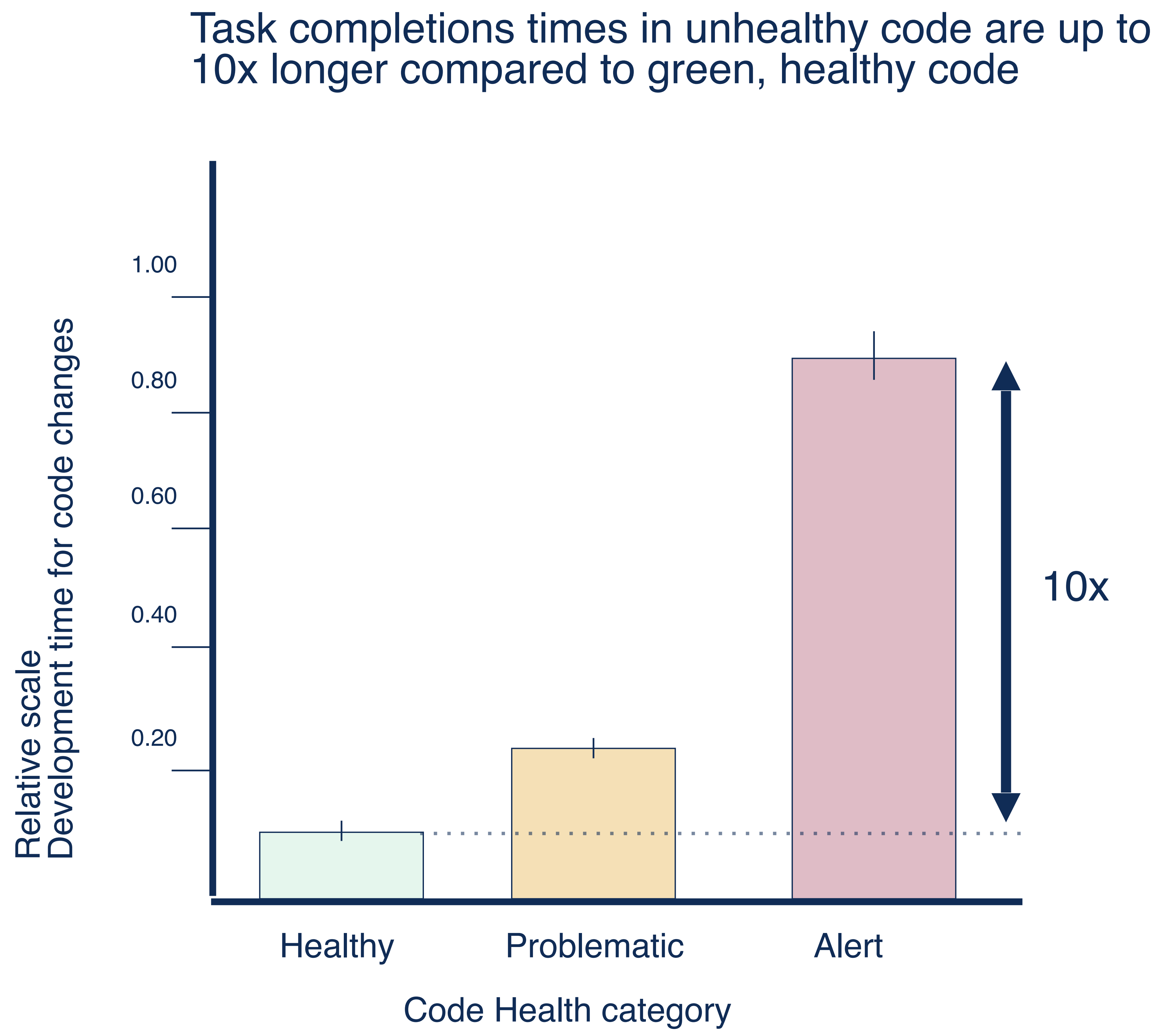
Are we outsourcing the fun and adding to the mundane?

> “We’ve all turned ourselves into maintenance programmers; we took the fun bit and we’re just going to give ourselves code that somebody else wrote.”

Kevlin Henney, 2024

The bigger picture

Yes, it’s possible to bring that +55% to 10X



Tornhill, A. & Borg, M. (2022) Code Red: The Business Impact of Code Quality https://arxiv.org/abs/2203.04374

AI accelerates the creation of new code — code quality is more important than ever!

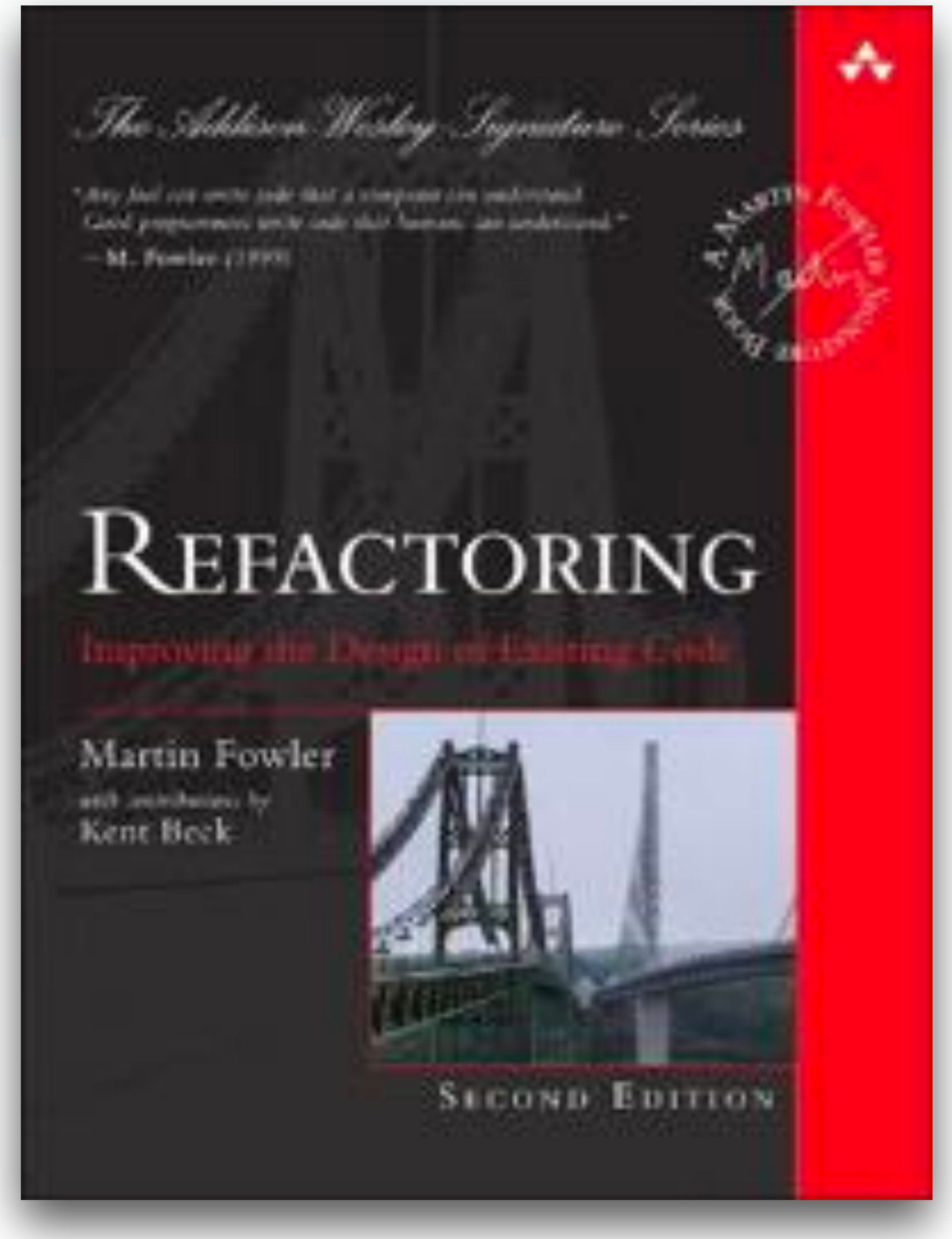
Yellow & Red Code comes with a significant on-boarding cost:

as a newcomer, you need

- **45% more time** for small tasks, and
- **93% more time** for large tasks compared to Green Code.

Borg, M., Tornhill, A., & Mones, E. (2023). U Owns the Code That Changes and How Marginal Owners Resolve Issues Slower in Low-Quality Source Code: https://arxiv.org/pdf/2304.11636.pdf

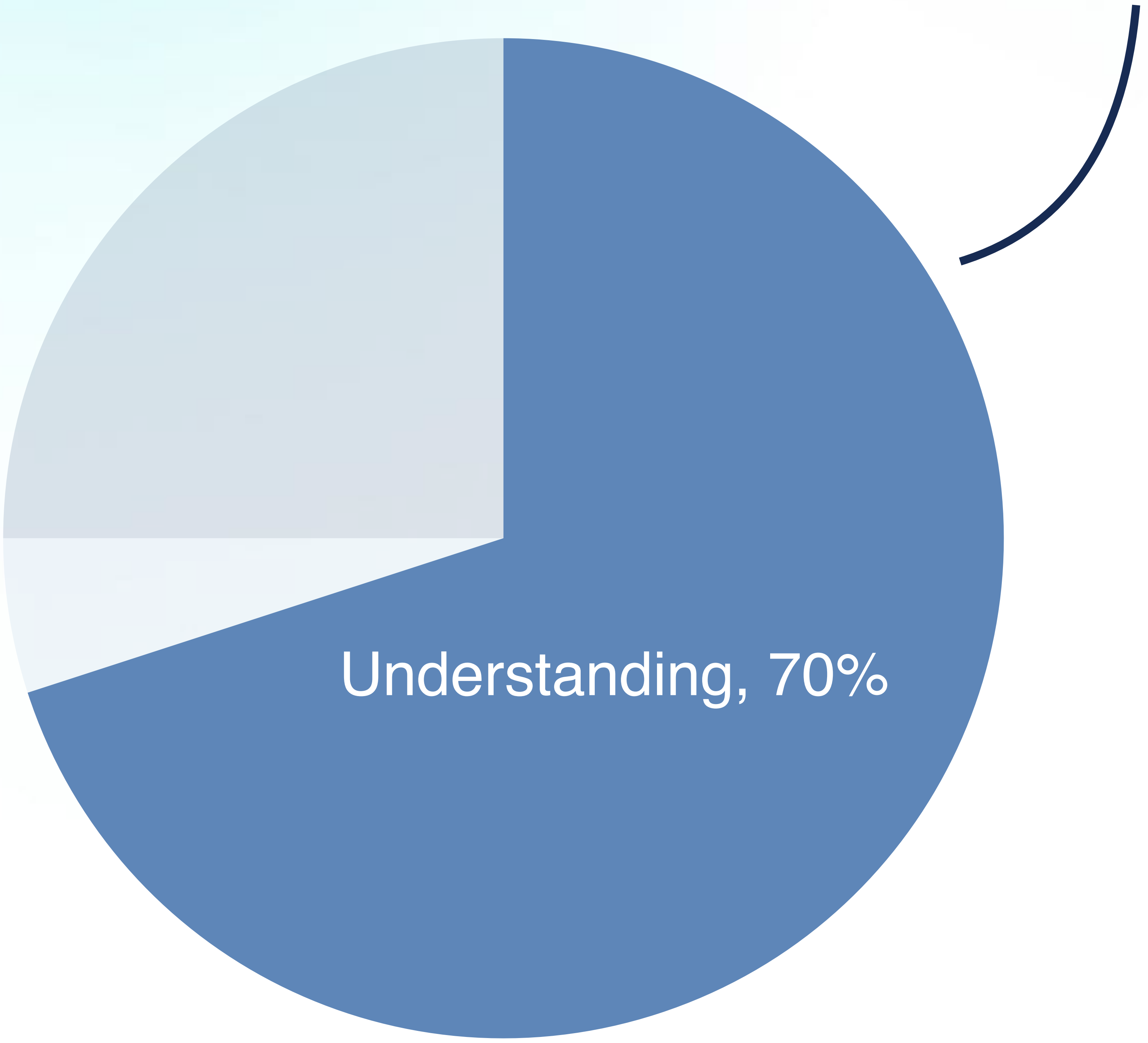
Refactoring and refuctoring


The Addison Wesley Signature Series
Refactoring
Improving the Design of Existing Code
Martin Fowler
with contributions by
Kent Beck
Second Edition

Refactoring is defined as **improving the design of existing code without changing its behavior**.

✓ It’s not a refactoring unless we improve the design.

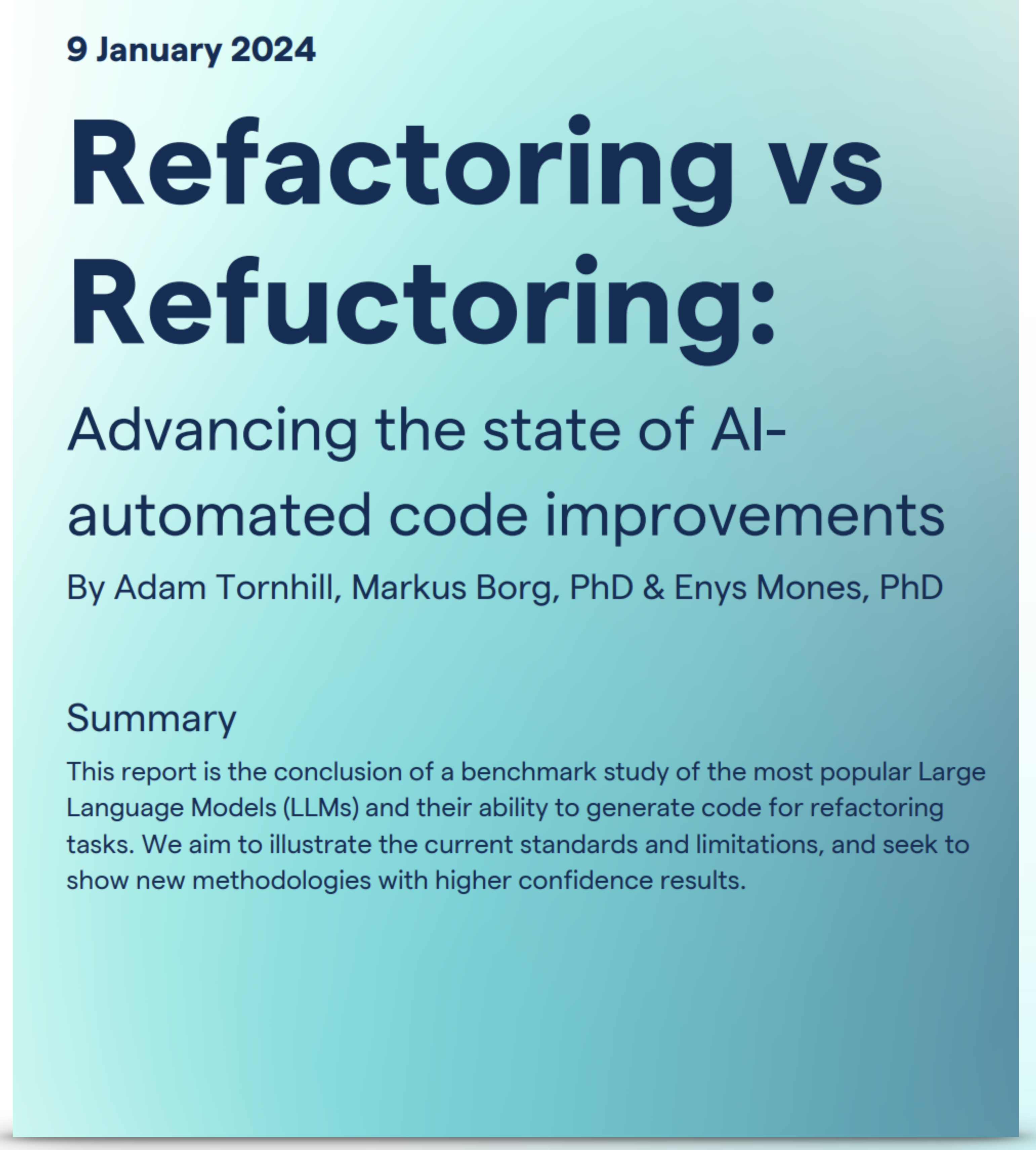
✓ It’s not a refactoring if we fail to preserve the behavior of the original code, e.g. we introduce a bug.



Refactoring improves this part

⚠️ Refuctoring: the process of changing existing code while – involuntarily – altering the program’s behavior

[Research:] **Let’s use AI to automate refactoring**



9 January 2024

Refactoring vs Refuctoring:

Advancing the state of AI-automated code improvements

By Adam Tornhill, Markus Borg, PhD & Enys Mones, PhD

Summary

This report is the conclusion of a benchmark study of the most popular Large Language Models (LLMs) and their ability to generate code for refactoring tasks. We aim to illustrate the current standards and limitations, and seek to show new methodologies with higher confidence results.

100k+ refactorings generated with AI

Open source Javascript and Typescript

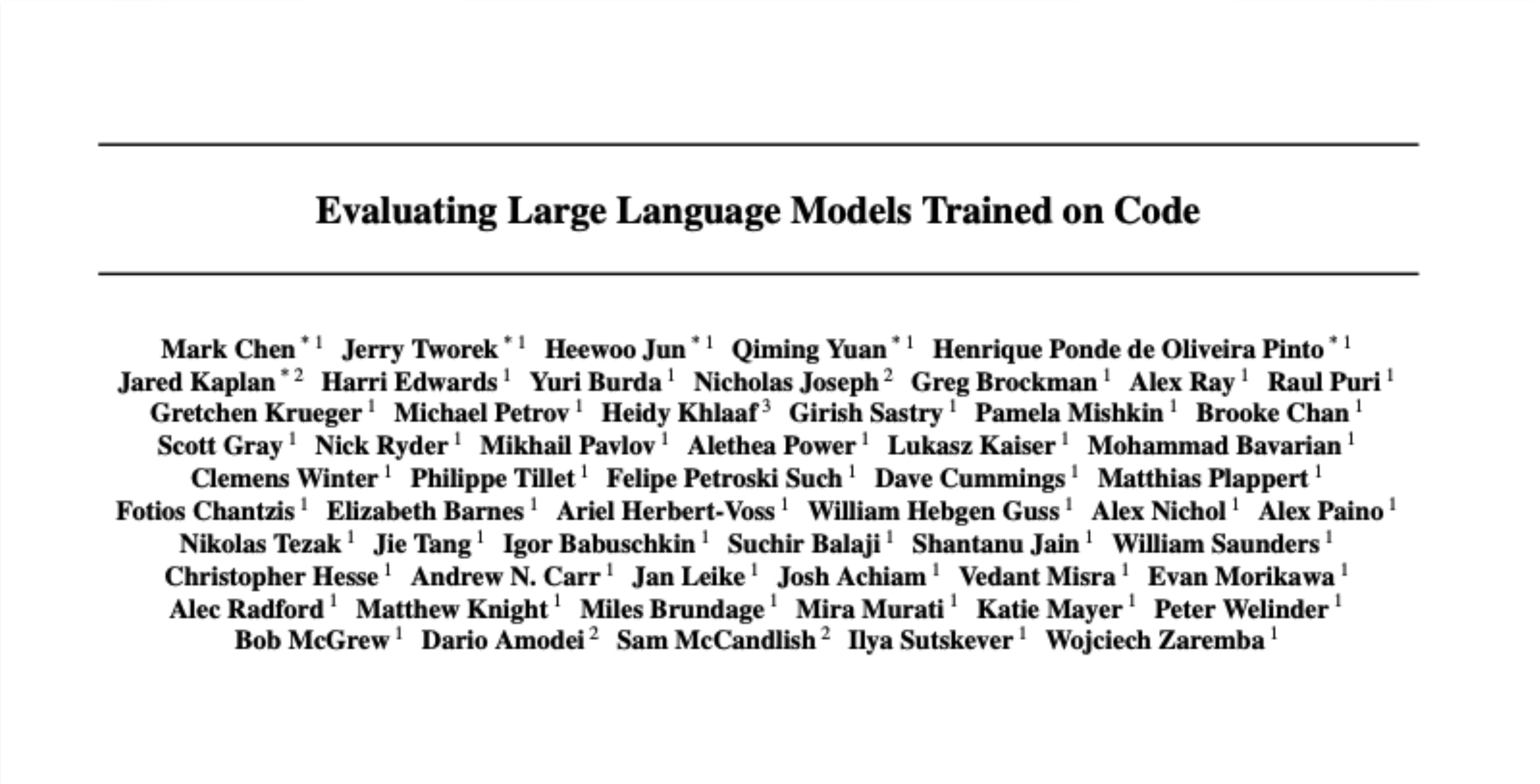
Benchmarking criteria: Code Health as the gold standard for code improvements

[1] https://codescene.io/docs/guides/technical/code-health.html

[Research:] **Can AI help us improve existing code?**

AI model	**Valid code?** (check the syntax of the refactored code)	**Code Health improved?** (did the code change by the AI mitigate the code smell?)	**Valid refactoring?** (do the tests still pass after the AI changed the code?)
PaLM 2 code [Google]	**99.93%**	**68.75%**	**32.29%**
GPT 3.5 [OpenAI]	**100%**	**69.89%**	**30.26%**
PaLM 2 text [Google]	**100%**	**66.54%**	**34.73%**
phind-codellama-34b-v2 [Meta, Phind]	**100%**	**78.76%**	**18.14%**

The average code quality



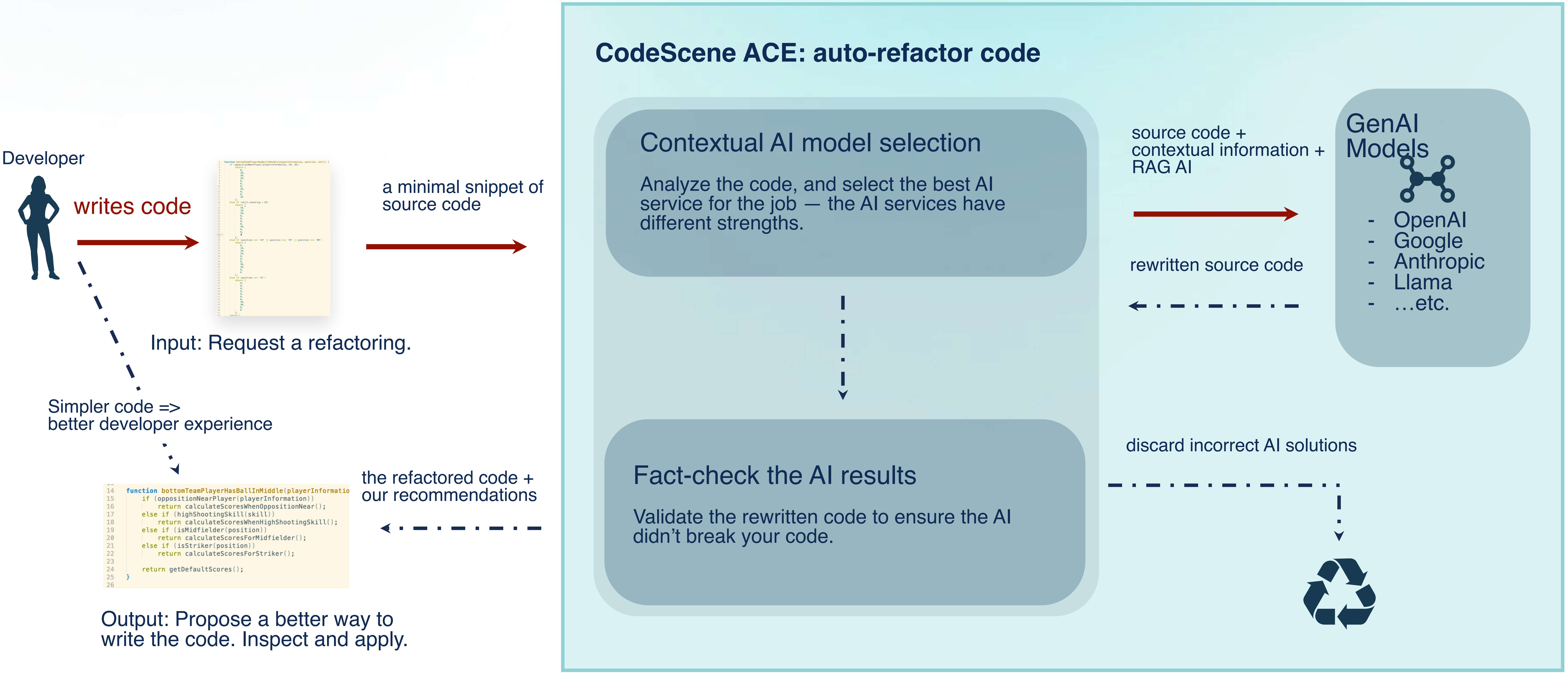
Evaluating Large Language Models Trained on Code

Mark Chen[*1] Jerry Tworek[*1] Heewoo Jun[*1] Qiming Yuan[*1] Henrique Ponde de Oliveira Pinto[*1]
Jared Kaplan[*2] Harri Edwards[1] Yuri Burda[1] Nicholas Joseph[2] Greg Brockman[1] Alex Ray[1] Raul Puri[1]
Gretchen Krueger[1] Michael Petrov[1] Heidy Khlaaf[3] Girish Sastry[1] Pamela Mishkin[1] Brooke Chan[1]
Scott Gray[1] Nick Ryder[1] Mikhail Pavlov[1] Alethea Power[1] Lukasz Kaiser[1] Mohammad Bavarian[1]
Clemens Winter[1] Philippe Tillet[1] Felipe Petroski Such[1] Dave Cummings[1] Matthias Plappert[1]
Fotios Chantzis[1] Elizabeth Barnes[1] Ariel Herbert-Voss[1] William Hebgen Guss[1] Alex Nichol[1] Alex Paino[1]
Nikolas Tezak[1] Jie Tang[1] Igor Babuschkin[1] Suchir Balaji[1] Shantanu Jain[1] William Saunders[1]
Christopher Hesse[1] Andrew N. Carr[1] Jan Leike[1] Josh Achiam[1] Vedant Misra[1] Evan Morikawa[1]
Alec Radford[1] Matthew Knight[1] Miles Brundage[1] Mira Murati[1] Katie Mayer[1] Peter Welinder[1]
Bob McGrew[1] Dario Amodei[2] Sam McCandlish[2] Ilya Sutskever[1] Wojciech Zaremba[1]

"We believe this is unlikely to be a large factor here, as **the GitHub dataset contains plenty of poor-quality code**.

The bugs are designed to be of the sort we'd expect to appear commonly in the dataset; code that compiles and often runs without errors but gives an incorrect answer."

[Innovation:] **Fact-checking the AI refactorings**



Developer

writes code

Input: Request a refactoring.

a minimal snippet of source code

Simpler code => better developer experience

```
function bottomTeamPlayerHasBallInMiddle(playerInformatio
    if (oppositionNearPlayer(playerInformation))
        return calculateScoresWhenOppositionNear();
    else if (highShootingSkill(skill))
        return calculateScoresWhenHighShootingSkill();
    else if (isMidfielder(position))
        return calculateScoresForMidfielder();
    else if (isStriker(position))
        return calculateScoresForStriker();

    return getDefaultScores();
}
```

the refactored code + our recommendations

Output: Propose a better way to write the code. Inspect and apply.

CodeScene ACE: auto-refactor code

Contextual AI model selection

Analyze the code, and select the best AI service for the job — the AI services have different strengths.

Fact-check the AI results

Validate the rewritten code to ensure the AI didn't break your code.

source code + contextual information + RAG AI

rewritten source code

discard incorrect AI solutions

GenAI Models

- OpenAI
- Google
- Anthropic
- Llama
- ...etc.

[Outcome:] **Elevate AI to the level of human experts with a fact-checking model**

	Complex Conditional	Deep, Nested Complexity	Bumpy Road	Complex Method
Raw GPT-3.5	33.7%	26.0%	26.3%	28.2%
GPT-3.5 with fact-checking	96.7%	98.4%	97.8%	98.9%

CodeScene ACE combines the results of multiple AIs and reject the incorrect solutions, **98%** of the remaining AI-generated refactorings improve the code without breaking it.

With fact-checking, we can elevate generative AI to achieve 10X

Q&A

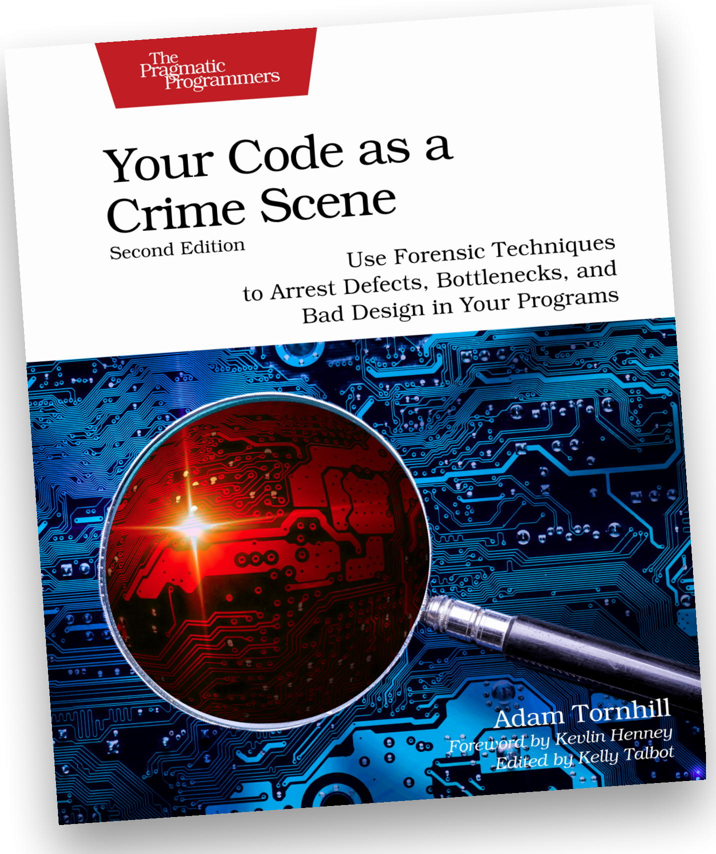
References

Code Red: The business impact of code quality

- https://codescene.com/hubfs/web_docs/Business-impact-of-low-code-quality.pdf

Refactoring vs Refuctoring: Advancing the state of AI automated code improvements

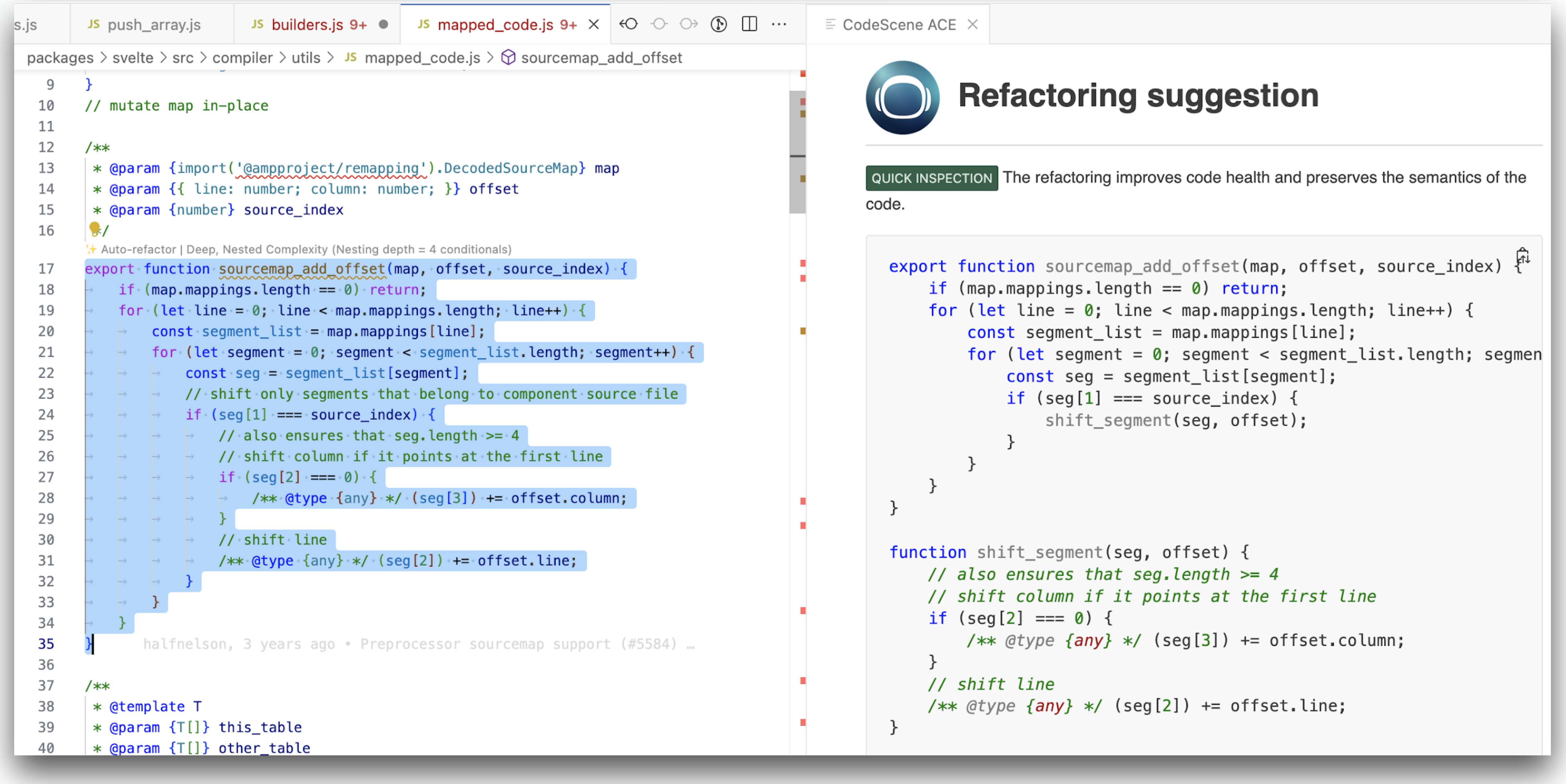
- https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf


The Pragmatic Programmers
Your Code as a Crime Scene
Second Edition
Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs
Adam Tornhill
Foreword by Kevin Henney
Edited by Kelly Talbot

Your Code as a Crime Scene, 2nd ed (2023)

https://twitter.com/AdamTornhill

[Free] Try the automated refactoring via CodeScene


s.js
push_array.js
builders.js 9+
mapped_code.js 9+
CodeScene ACE
packages > svelte > src > compiler > utils > mapped_code.js > sourcemap_add_offset

```
}
// mutate map in-place

/**
 * @param {import('@ampproject/remapping').DecodedSourceMap} map
 * @param {{ line: number; column: number; }} offset
 * @param {number} source_index
 */
✨ Auto-refactor | Deep, Nested Complexity (Nesting depth = 4 conditionals)
export function sourcemap_add_offset(map, offset, source_index) {
    if (map.mappings.length == 0) return;
    for (let line = 0; line < map.mappings.length; line++) {
        const segment_list = map.mappings[line];
        for (let segment = 0; segment < segment_list.length; segment++) {
            const seg = segment_list[segment];
            // shift only segments that belong to component source file
            if (seg[1] === source_index) {
                // also ensures that seg.length >= 4
                // shift column if it points at the first line
                if (seg[2] === 0) {
                    /** @type {any} */ (seg[3]) += offset.column;
                }
                // shift line
                /** @type {any} */ (seg[2]) += offset.line;
            }
        }
    }
}       halfnelson, 3 years ago • Preprocessor sourcemap support (#5584) …

/**
 * @template T
 * @param {T[]} this_table
 * @param {T[]} other_table
```

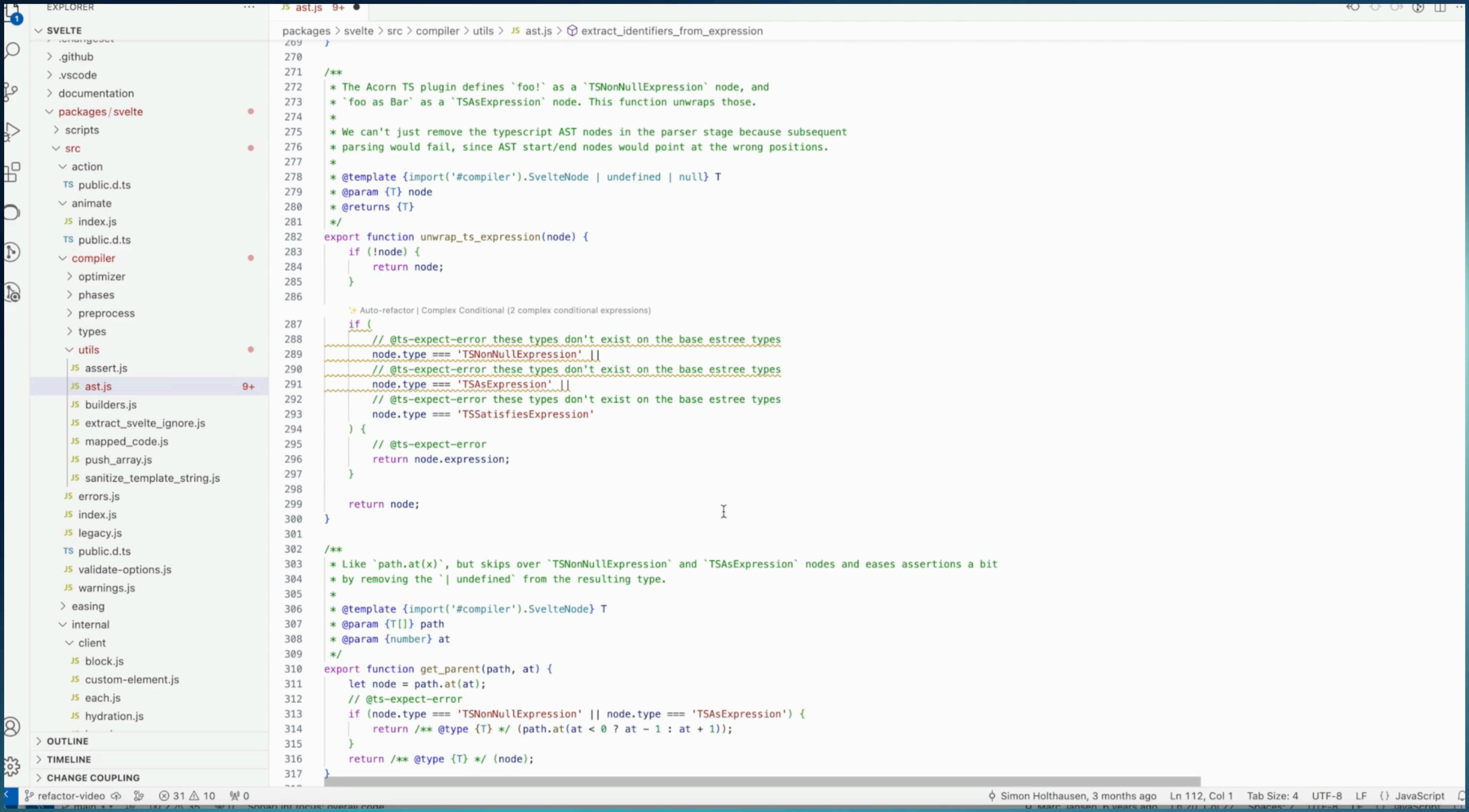
Refactoring suggestion

QUICK INSPECTION The refactoring improves code health and preserves the semantics of the code.

```
export function sourcemap_add_offset(map, offset, source_index) {
    if (map.mappings.length == 0) return;
    for (let line = 0; line < map.mappings.length; line++) {
        const segment_list = map.mappings[line];
        for (let segment = 0; segment < segment_list.length; segmen
            const seg = segment_list[segment];
            if (seg[1] === source_index) {
                shift_segment(seg, offset);
            }
        }
    }
}

function shift_segment(seg, offset) {
    // also ensures that seg.length >= 4
    // shift column if it points at the first line
    if (seg[2] === 0) {
        /** @type {any} */ (seg[3]) += offset.column;
    }
    // shift line
    /** @type {any} */ (seg[2]) += offset.line;
}
```

Demo



```
/**
 * The Acorn TS plugin defines `foo!` as a `TSNonNullExpression` node, and
 * `foo as Bar` as a `TSAsExpression` node. This function unwraps those.
 *
 * We can't just remove the typescript AST nodes in the parser stage because subsequent
 * parsing would fail, since AST start/end nodes would point at the wrong positions.
 *
 * @template {import('#compiler').SvelteNode | undefined | null} T
 * @param {T} node
 * @returns {T}
 */
export function unwrap_ts_expression(node) {
	if (!node) {
		return node;
	}

	✨ Auto-refactor | Complex Conditional (2 complex conditional expressions)
	if (
		// @ts-expect-error these types don't exist on the base estree types
		node.type === 'TSNonNullExpression' ||
		// @ts-expect-error these types don't exist on the base estree types
		node.type === 'TSAsExpression' ||
		// @ts-expect-error these types don't exist on the base estree types
		node.type === 'TSSatisfiesExpression'
	) {
		// @ts-expect-error
		return node.expression;
	}

	return node;
}

/**
 * Like `path.at(x)`, but skips over `TSNonNullExpression` and `TSAsExpression` nodes and eases assertions a bit
 * by removing the `| undefined` from the resulting type.
 *
 * @template {import('#compiler').SvelteNode} T
 * @param {T[]} path
 * @param {number} at
 */
export function get_parent(path, at) {
	let node = path.at(at);
	// @ts-expect-error
	if (node.type === 'TSNonNullExpression' || node.type === 'TSAsExpression') {
		return /** @type {T} */ (path.at(at < 0 ? at - 1 : at + 1));
	}
	return /** @type {T} */ (node);
}
```

So, when I open up this code, I see that CodeScene identifies a couple of code smells for me.

codescene.com/ai

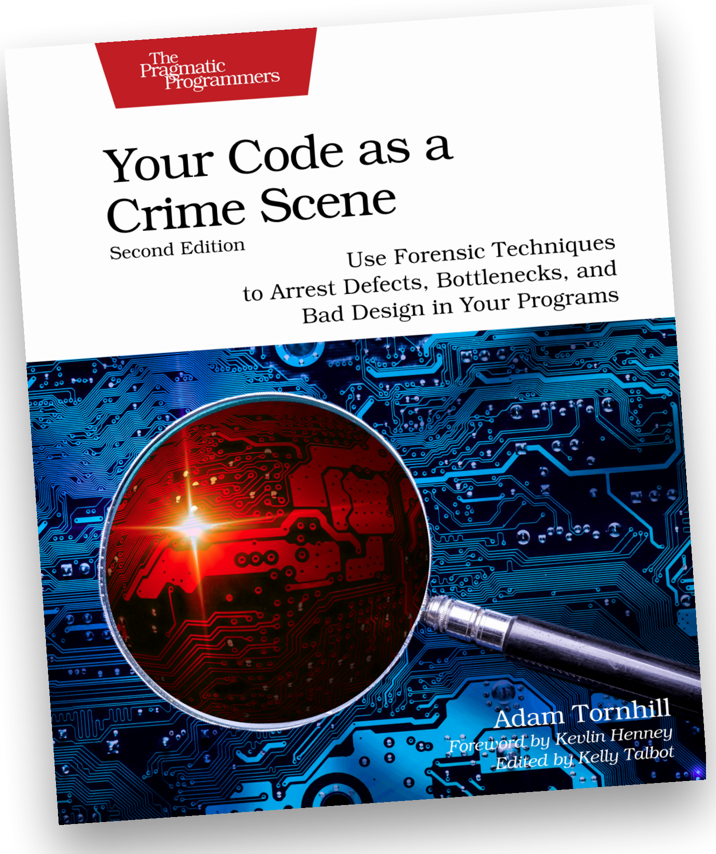
References

Code Red: The business impact of code quality

- https://codescene.com/hubfs/web_docs/Business-impact-of-low-code-quality.pdf

Refactoring vs Refuctoring: Advancing the state of AI automated code improvements

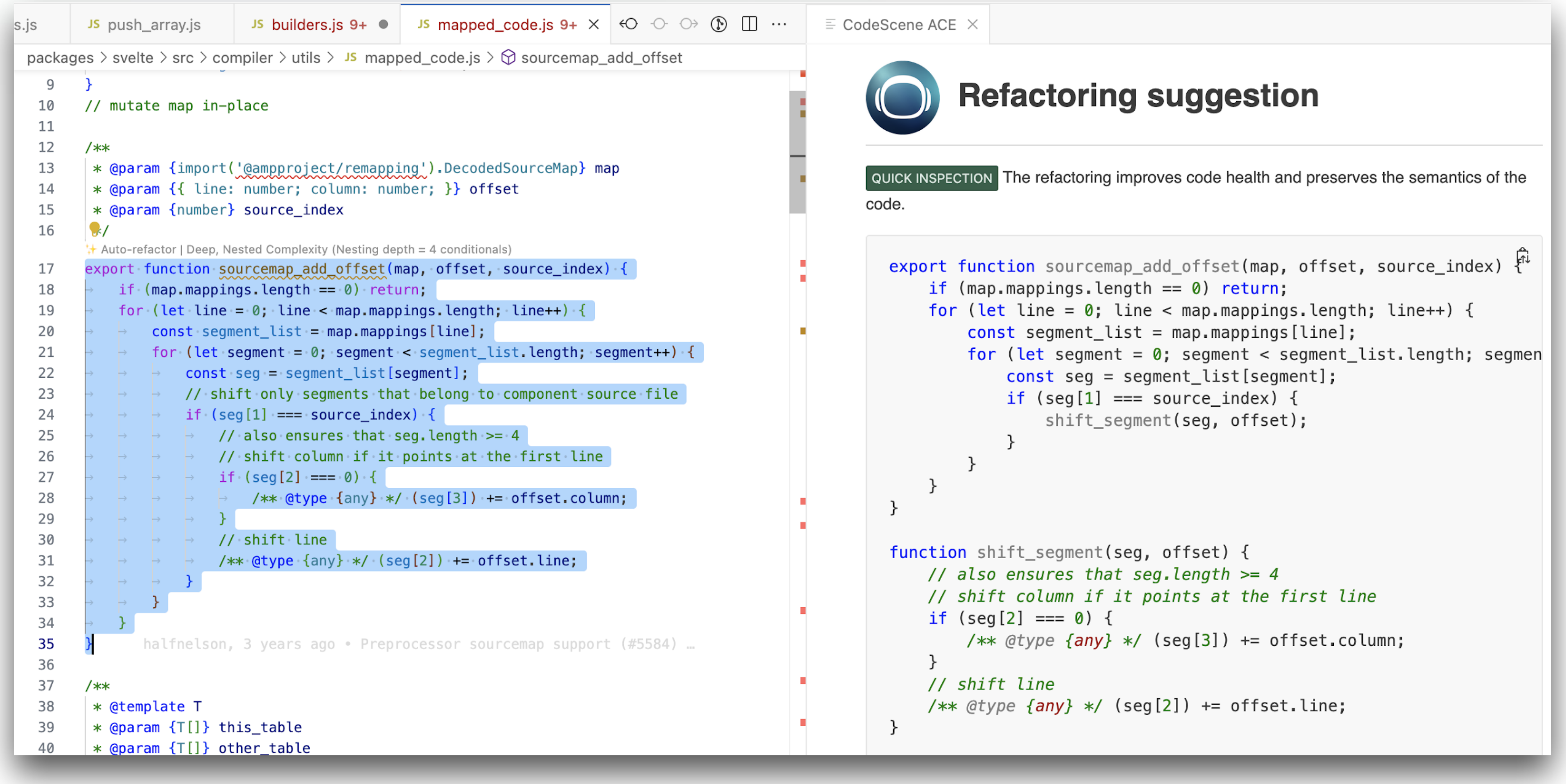
- https://codescene.com/hubfs/whitepapers/Refactoring-vs-Refuctoring-Advancing-the-state-of-AI-automated-code-improvements.pdf


The Pragmatic Programmers
Your Code as a Crime Scene
Second Edition
Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs
Adam Tornhill
Foreword by Kevin Henney
Edited by Kelly Talbot

Your Code as a Crime Scene, 2nd ed (2023)

https://twitter.com/AdamTornhill

[Free] Try the automated refactoring via CodeScene


s.js
push_array.js
builders.js 9+
mapped_code.js 9+
CodeScene ACE
packages > svelte > src > compiler > utils > mapped_code.js > sourcemap_add_offset

```
}
// mutate map in-place

/**
 * @param {import('@ampproject/remapping').DecodedSourceMap} map
 * @param {{ line: number; column: number; }} offset
 * @param {number} source_index
 */
✨ Auto-refactor | Deep, Nested Complexity (Nesting depth = 4 conditionals)
export function sourcemap_add_offset(map, offset, source_index) {
	if (map.mappings.length == 0) return;
	for (let line = 0; line < map.mappings.length; line++) {
		const segment_list = map.mappings[line];
		for (let segment = 0; segment < segment_list.length; segment++) {
			const seg = segment_list[segment];
			// shift only segments that belong to component source file
			if (seg[1] === source_index) {
				// also ensures that seg.length >= 4
				// shift column if it points at the first line
				if (seg[2] === 0) {
					/** @type {any} */ (seg[3]) += offset.column;
				}
				// shift line
				/** @type {any} */ (seg[2]) += offset.line;
			}
		}
	}
}        halfnelson, 3 years ago • Preprocessor sourcemap support (#5584) …

/**
 * @template T
 * @param {T[]} this_table
 * @param {T[]} other_table
```

Refactoring suggestion

QUICK INSPECTION The refactoring improves code health and preserves the semantics of the code.

```
export function sourcemap_add_offset(map, offset, source_index) {
    if (map.mappings.length == 0) return;
    for (let line = 0; line < map.mappings.length; line++) {
        const segment_list = map.mappings[line];
        for (let segment = 0; segment < segment_list.length; segmen
            const seg = segment_list[segment];
            if (seg[1] === source_index) {
                shift_segment(seg, offset);
            }
        }
    }
}

function shift_segment(seg, offset) {
    // also ensures that seg.length >= 4
    // shift column if it points at the first line
    if (seg[2] === 0) {
        /** @type {any} */ (seg[3]) += offset.column;
    }
    // shift line
    /** @type {any} */ (seg[2]) += offset.line;
}
```

FEEDBACK

Adam

- What is the selling point? - Audience is managers, they want their code quality under control
- Don't: CS does this or that / Do: this is how code health looks like, etc (don't try to sell CS)
- Eyeopener: Video on code health of an actual code base from Adam
- Focus on: Be much more condensed (more iterations needed!)
- Have some text that the audience can rest their eyes on.
- Max 1 min per slide - pick a few lines you use to explain the things

- Code health transition too rough - audience might think "why should we care about it"?
- Quote Adam's book: ""

FEEDBACK

Lydia

- Took too long to get to the AI part