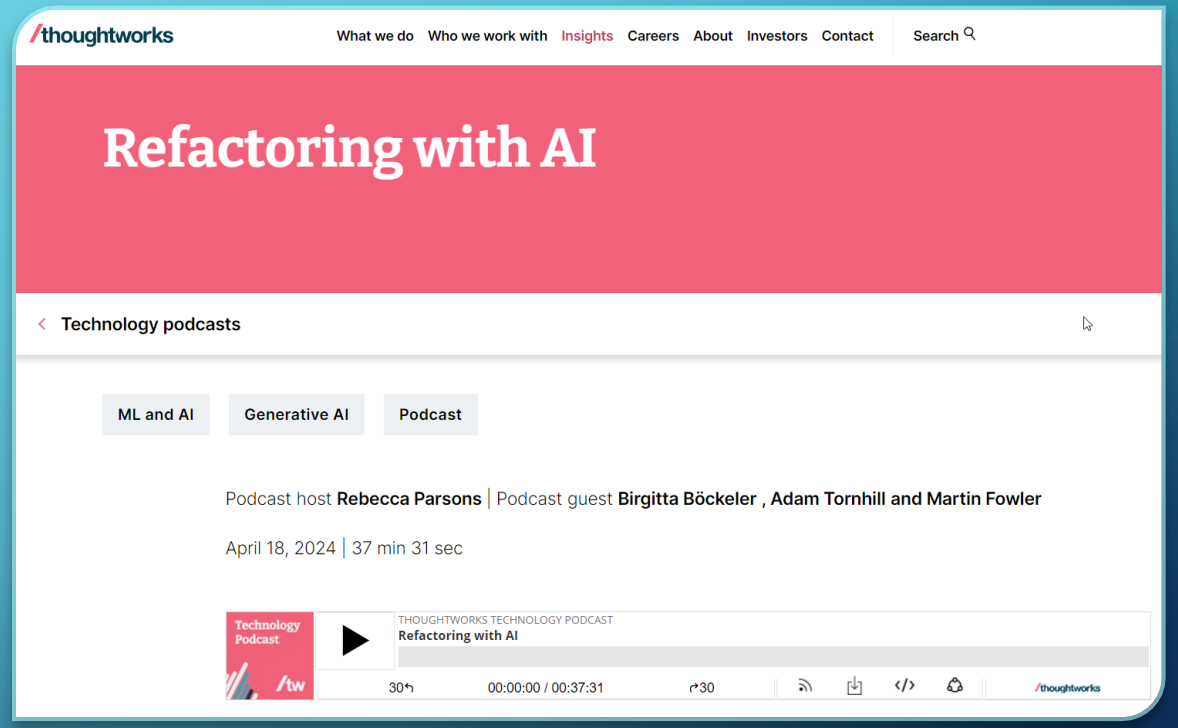
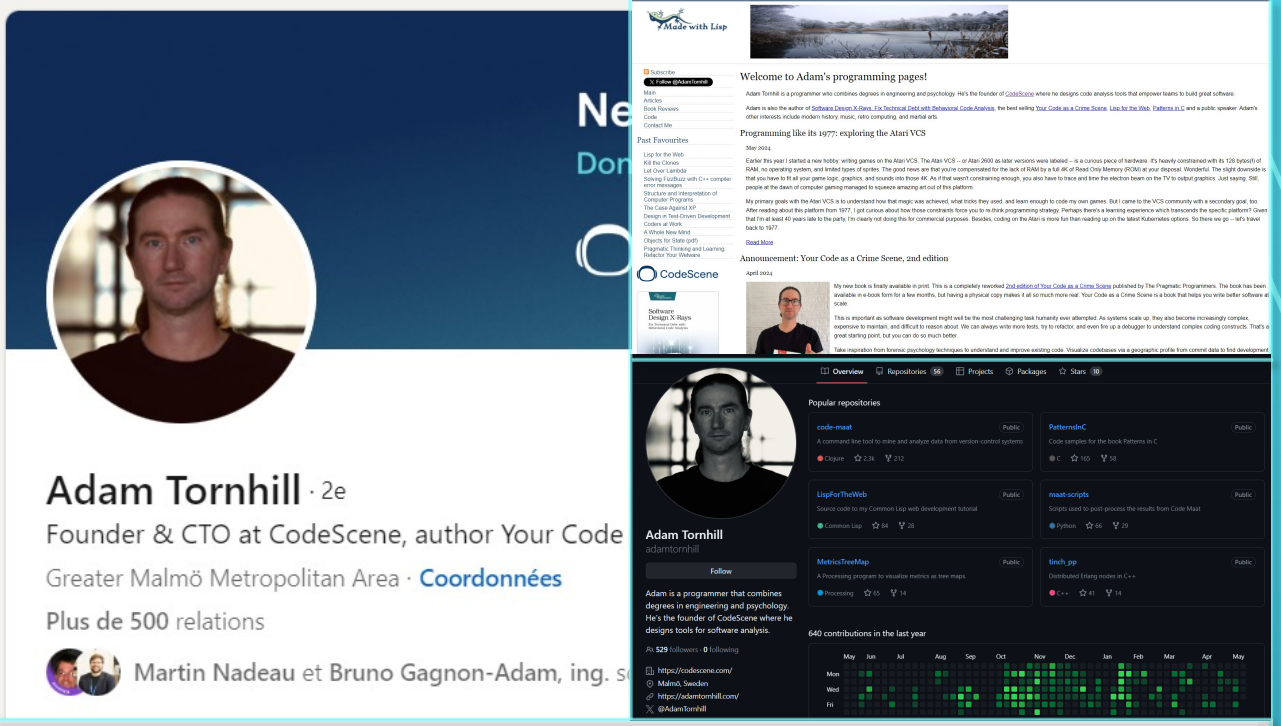
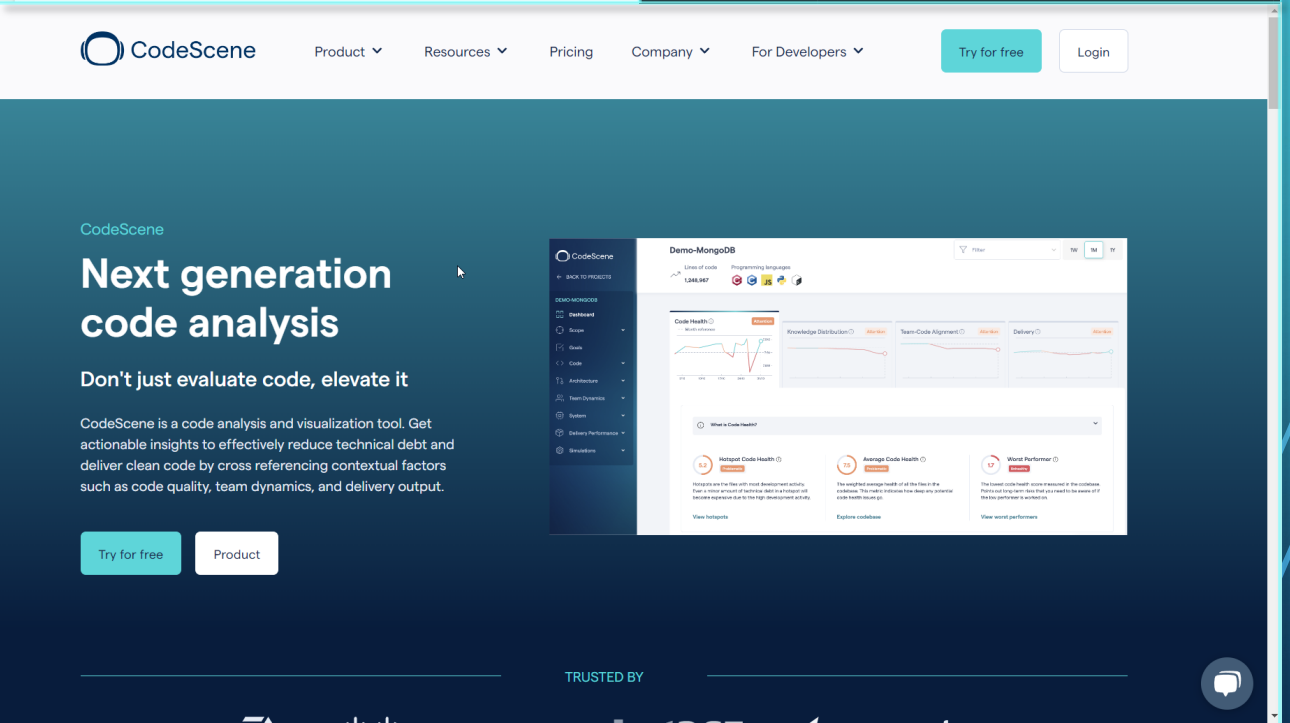
REFACTORING WITH AI

BASED AFTER PODCAST FROM THOUGHTWORKS

- PodCast from Thoughtworks :
- Discussion between known experts in domain, sparked by close attention will AI replace soon all coders out there.
- Main guest of Podcast - Adam Tornhill



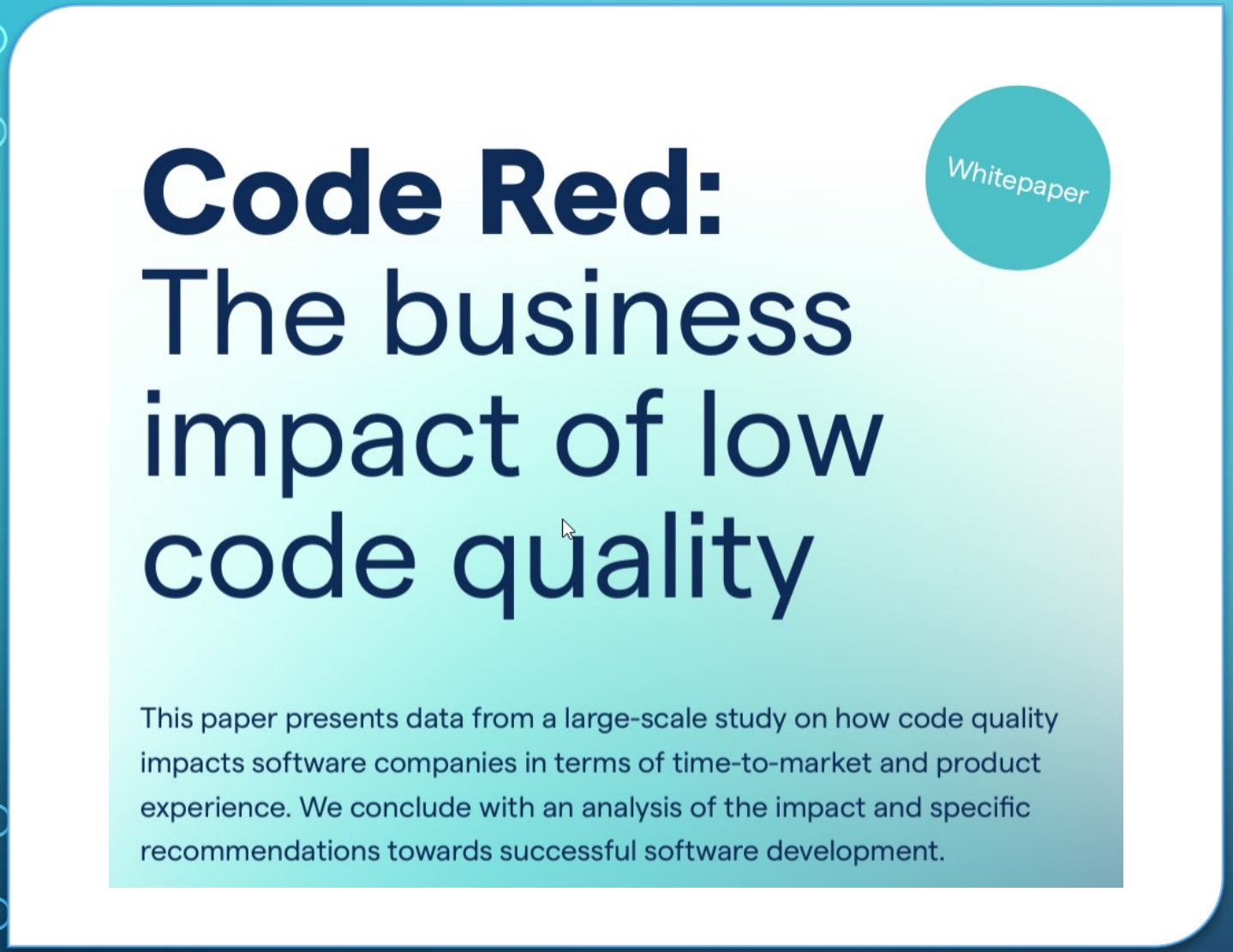




ADAM TORNHILL

- Author of few books and articles
  - Your Code as a Crime Scene
  - Software Design X-Rays. Fix Technical Debt with Behavioral Code Analysis
  - Increasing, not Diminishing: Investigating the Returns of Highly Maintainable Code. Why code quality matters.

CODE QUALITY, WHY DOES IT MATTER



KEY TAKEAWAYS

Everyone in the software industry "knows" that code quality is important, yet we never had any data or numbers to prove it. Consequently, the importance of a healthy codebase is largely undervalued at the business level.
With this paper we remove the quotation marks so that "knows" becomes **knows** by attaching numbers on the impact of unhealthy code. That way, code quality can finally become the business concern that this study shows that it must be.

- Efficient software development is a competitive advantage that enables companies to maintain a short time-to-market with a mature product experience.
- However, research shows that 23-42% of developer's time is wasted due to Technical Debt and bad code.
- A key reason that this waste is tolerated is because code quality lacks visibility to non-tech stakeholders and possible gains in code quality are hard to translate into business value.
- This paper aims to elevate code quality to the business level by putting numbers on the impact of unhealthy code.
- Our research investigates 39 commercial codebases from various industries and domains. The finding are peer reviewed, statistically significant, and reproducible.[1] All metrics were automated via CodeScene.
- The results show that code quality has a dramatic impact on, both, time-to-market as well as the external quality of the product. High quality code has:
  A. 15 times fewer bugs,
  B. twice the development speed, and
  C. 9 times lower uncertainty in completion time.

REFACTORING VS REFUCTORING

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.

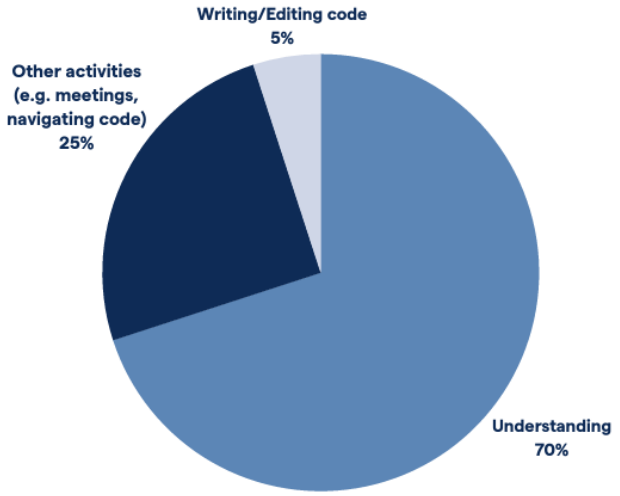


Figure 2: The majority of a developer's time is spent trying to understand the existing system (data from Minelli, et. al., 2015) 1

1. https://ieeexplore.ieee.org/abstract/document/7181430

RAW USAGE

Using this data, we measure the ratio of refactoring vs refuctoring for a series of popular AI models:

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	99.93%	68.75%	37.29%
GPT-3.5	100%	69.89%	30.26%
PaLM 2t	100%	66.54%	34.73%
phind-codellama-34B-v2*	100%	78.76%	18.14%

CODESCENE AI FACT CHECKING TECHNOLOGY

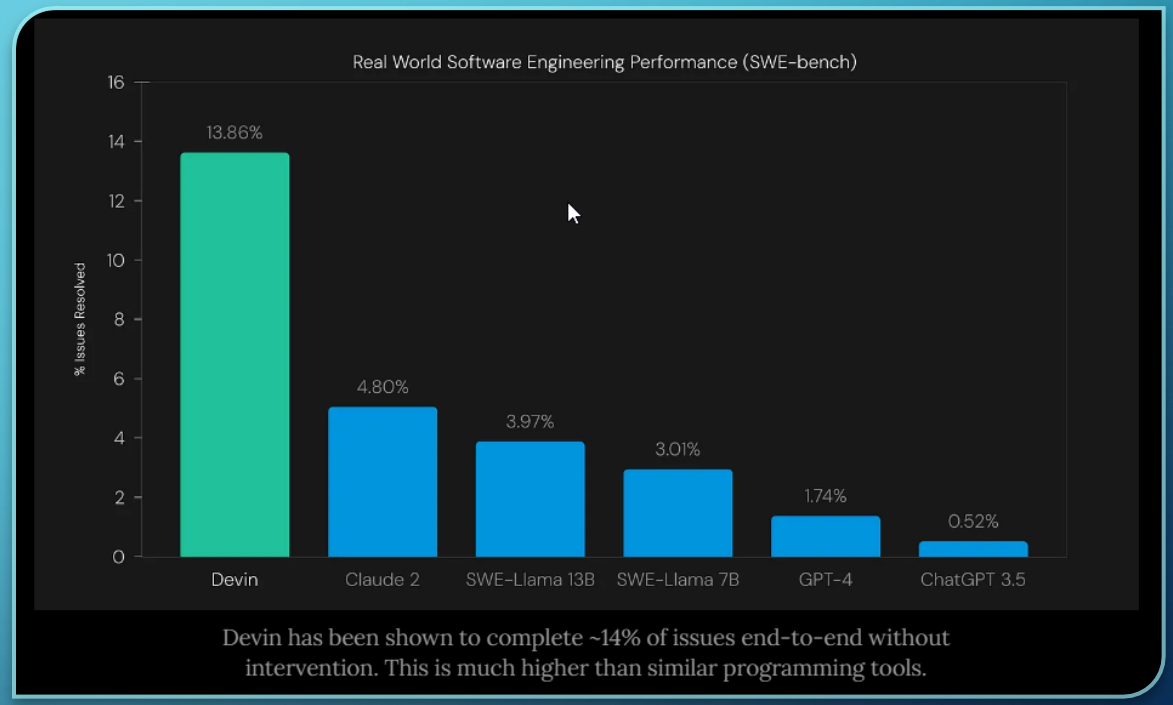
Benchmarking: improving AI correctness with a fact-checking model

To evaluate the fact-checking model, we re-ran the benchmarking study described in Table 1 above. This makes it possible to compare the correctness gained from the fact-checking model:

	Correct code smell refactorings			
Solution	**Complex Conditional**	**Deep Nested Logic**	**Bumpy Road**	**Complex Method**
Raw GPT-3.5	33.7%	26.0%	26.3%	28.2%
AI with CodeScene's fact-checking	96.7%	98.4%	97.8%	98.9%

GEN AI FOR CODE GENERATION

- Most of that code is not as good as authors of Gen-AI pretending it is.
- Devin – first AI software engineer
- Research found that using Copilot is strongly correlated with mistake-prone code being pushed to repositories. The code generated in 2023 resembled that of an "itinerant contributor, prone to violate the DRY-ness (Don't Repeat Yourself principle) of the repos visited"
- Copilot tended to generate more code than necessary without suggestions for refactoring or deleting unnecessary code



From blog of LOGAN THORNELOE

CONCLUSION

- This benchmarking study shows that AI is nowhere near replacing humans in a coding context; today's AI is simply too error-prone, and far from a point where it is able to securely modify existing code. However, by introducing a novel fact-checking model for the AI output, we can elevate generative AI to a point where it is genuinely useful as several complex code smells can be mitigated safely. This allows us to optimize for understanding – the dominant and most human-intensive aspect – not just the narrow task of writing new code.