CodeScene

Code Red: The business impact of code quality

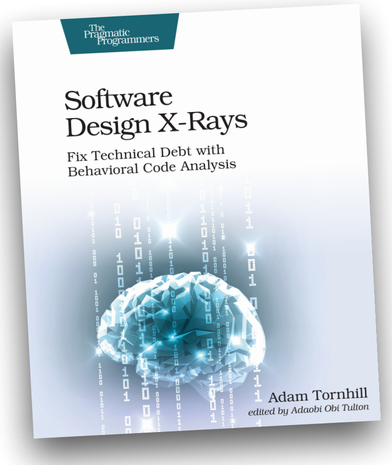
10 years of trauma & research in technical debt

November 2022

@AdamTornhill

codescene.com

What is Technical Debt?



“Technical debt is code that’s more expensive to maintain than it should be.”

Software Design X-Rays, 2018

What we actually know:

Research on Technical Debt

Waste

Software developers spend **23-42%** of their work week dealing with technical debt and bad code.[1, 2, 3]

Vulnerabilities

There is a statistically significant correlation between software vulnerabilities and code smells like Brain Classes, complex implementations, and large classes.[4]

[1] Besker, T., Martini, A., Bosch, J. (2019) “Software Developer Productivity Loss Due to Technical Debt”

[2] Stripe, (2018), “The Developer Coefficient: Software engineering efficiency and its $3 trillion impact on global GDP”

[3] https://codescene.com/technical-debt/whitepaper/calculate-business-costs-of-technical-debt.pdf

[4] Sultana, K. Z., Codabux, Z., & Williams, B. (2020, December). Examining the relationship of code and architectural smells with software vulnerabilities.

Technical Debt: where we are as an industry

Research finds that developers are **frequently forced to introduce new Technical Debt** as companies keep trading code quality for short-term gains like new features.[1]

[1] T Besker, A Martini, and J Bosch. 2019. “Software developer productivity loss due to technical debt—a replication and extension study examining 1207 developers’ development work”

Why short-term gains win over long-term maintainability:

Hyperbolic Discounting

> “There's never enough time to do something right, but there's always enough time to do it over.”*
>
> Melvin E. Conway (1968). “How Do Committees Invent?”

* Thanks, Kevlin Henney

Fighting hyperbolic discounting:

Visualise technical debt and code complexity

Visualize? How? Can we even measure “code quality” ?

“The assumption that fundamentally different views of complexity can be characterised by a single number is counter to the fundamental concepts of measurement theory.”

[..]

“the most promising approach is to identify specific attributes of complexity and measure these separately.”

Software Measurement: A Necessary Scientific Basis, N. Fenton (1994)

Code Health: an aggregated metric based on 25 factors

Source code

Parser

Examples on unhealthy code

Module level issues:

- **Low Cohesion**: many responsibilities
- **Brain Class**: low cohesion + large class + at least one Brain Method

Function level issues:

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

Implementation level issues:

- **Deeply Nested Logic:** if-statements inside if-statements
- **Primitive Obsession:** missing a domain language

Score, aggregate, and categorize

Code health categories:

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

Read more on how it works: https://codescene.com/blog/measure-code-health-of-your-codebase/

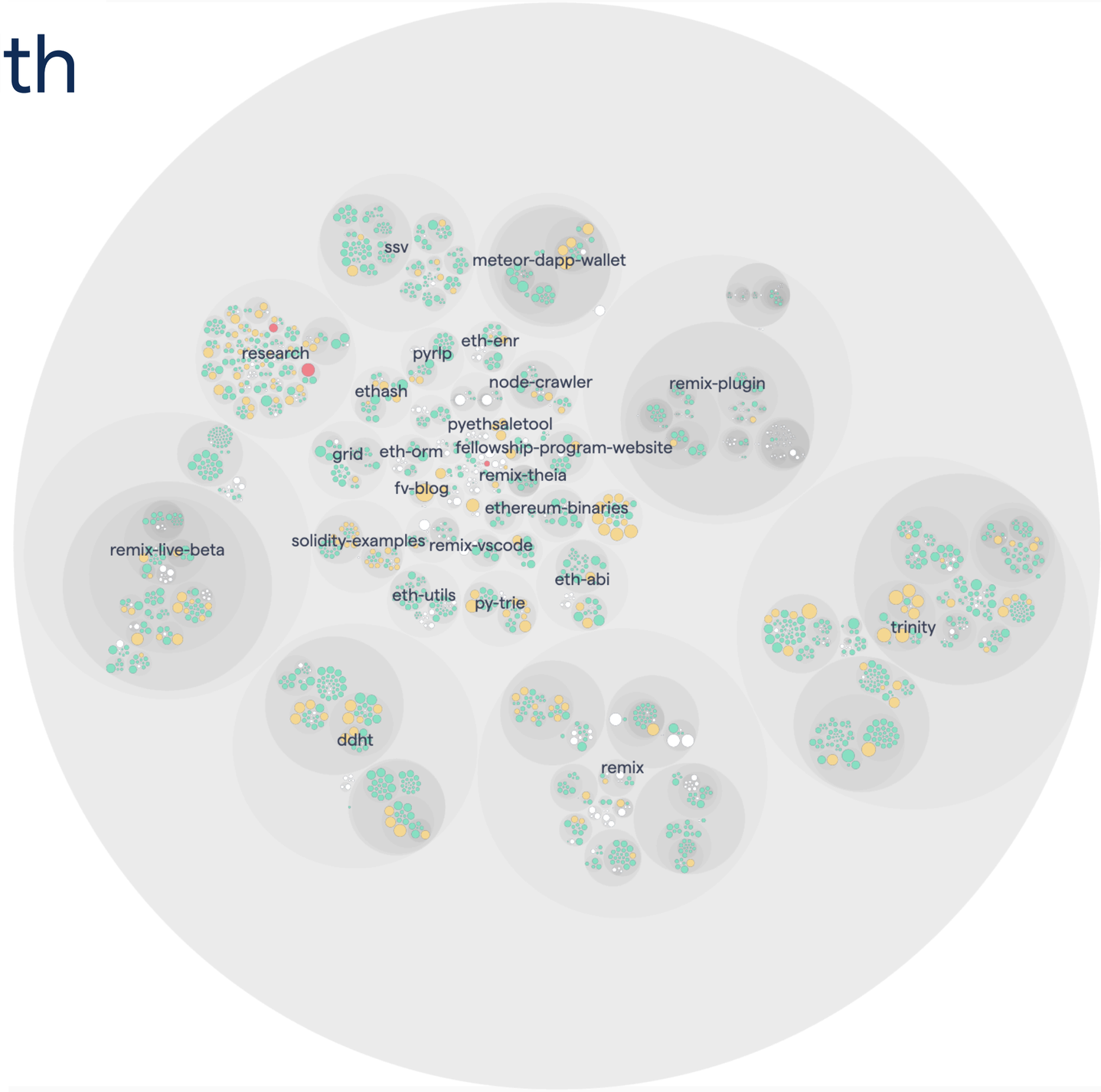
Visualizing code health

Ethereum: a decentralized, open-source blockchain with smart contract functionality

Example on 50 repositories

600k lines of code

https://github.com/ethereum



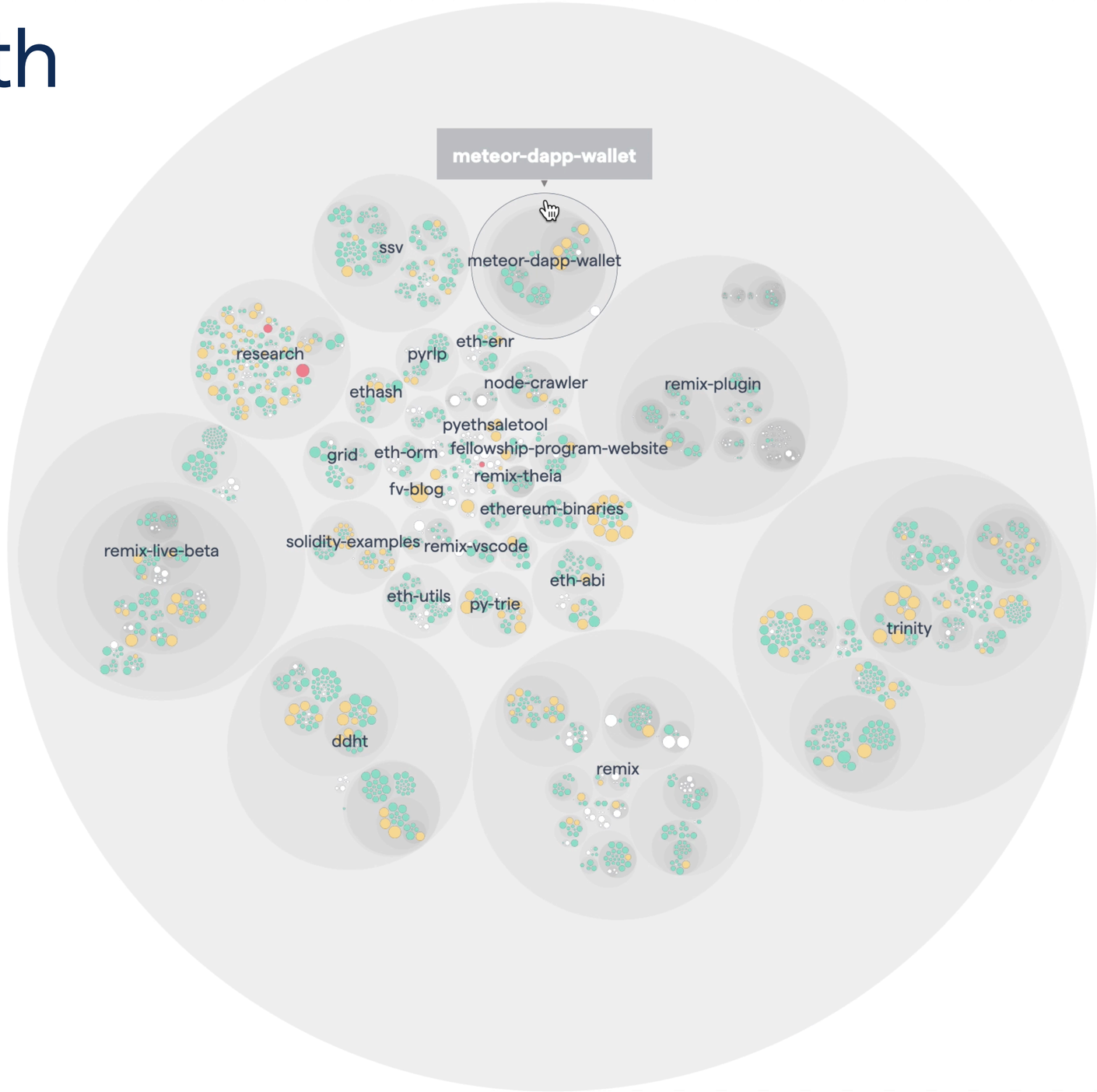
Visualizing code health

Ethereum: a decentralized, open-source blockchain with smart contract functionality

Example on 50 repositories

600k lines of code

https://github.com/ethereum



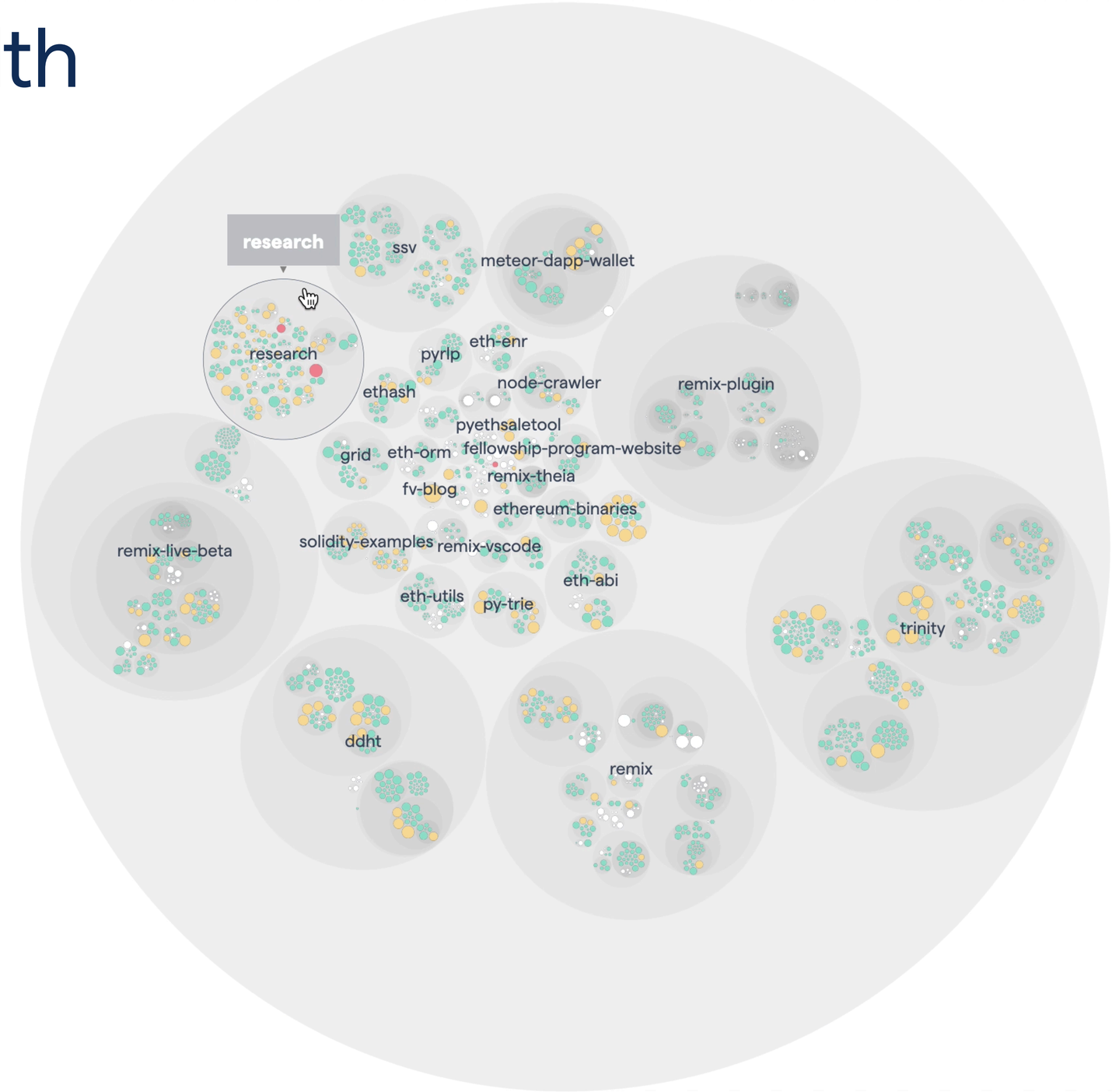
Visualizing code health

Ethereum: a decentralized, open-source blockchain with smart contract functionality

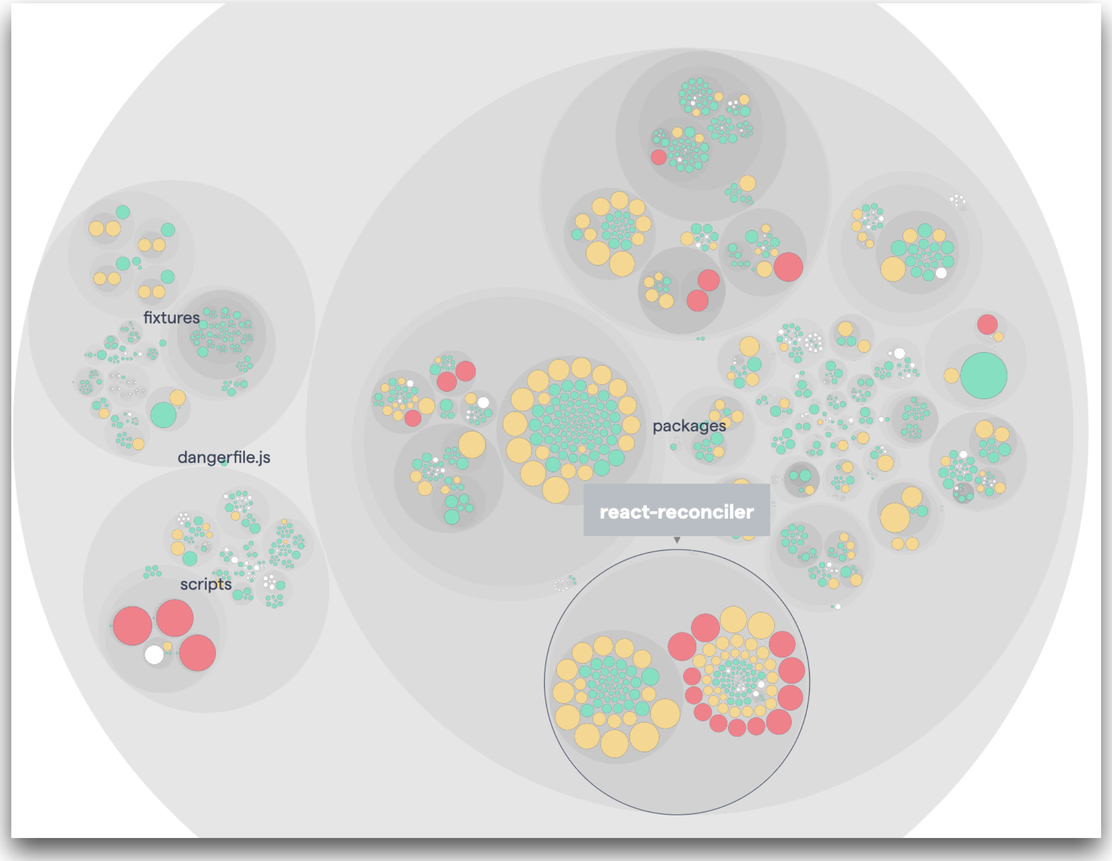
Example on 50 repositories

600k lines of code

https://github.com/ethereum



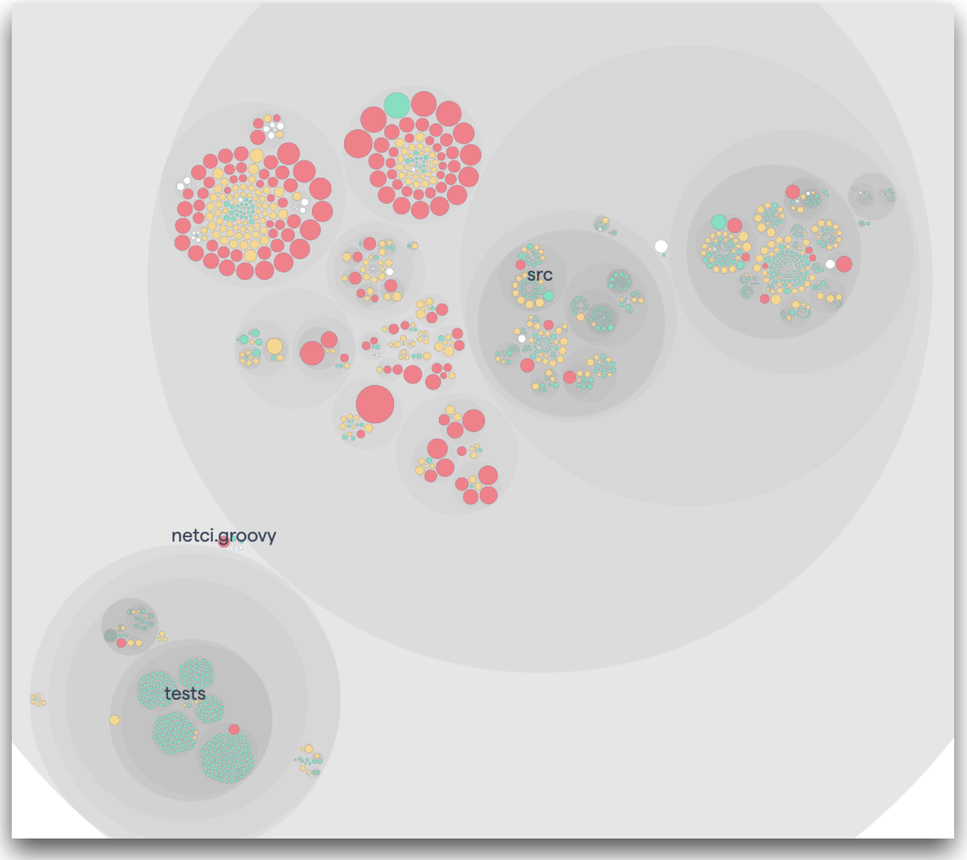
Examples: **a gallery of code**



React: a UI library

340k lines of code

https://github.com/facebook/react



CoreCLR: the runtime for .Net

8.5 million lines of code

https://github.com/dotnet/coreclr

From “knowing” to knowing:

Quantify the business impact of code quality

Research to quantify the impact of code quality: scope & data

- A quantitive large-scale study of code quality impact.
- Data from 39 commercial codebases.
- Analysed more than 40 000 software modules.
- Many different industry segments.
- Tested across 14 programming languages.
- Using the CodeScene tool to automated the analyses.
- Our research findings are statistical significant and peer reviewed for the International Conference on Technical Debt 2022[1]

Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases

Adam Tornhill
CodeScene
Malmö, Sweden
adam.tornhill@codescene.com

Markus Borg
RISE Research Institutes of Sweden
Lund University
Lund, Sweden
markus.borg@ri.se

ABSTRACT

Code quality remains an abstract concept that fails to get traction at the business level. Consequently, software companies keep trading code quality for time-to-market and new features. The resulting technical debt is estimated to waste up to 42% of developers' time. At the same time, there is a global shortage of software developers, meaning that developer productivity is key to software businesses. Our overall mission is to make code quality a business concern, not just a technical aspect. Our first goal is to understand how code quality impacts 1) the number of reported defects, 2) the time to resolve issues, and 3) the predictability of resolving issues on time. We analyze 39 proprietary production codebases from a variety of domains using the CodeScene tool based on a combination of source code analysis, version-control mining, and issue information from Jira. By analyzing activity in 30,737 files, we find that low quality code contains 15 times more defects than high quality code. Furthermore, resolving issues in low quality code takes on average 124% more time in development. Finally, we report that issue resolutions in low quality code involve higher uncertainty manifested as 9 times longer maximum cycle times. This study provides evidence that code quality cannot be dismissed as a technical concern. With 15 times fewer defects, twice the development speed, and substantially more predictable issue resolution times, the business advantage of high quality code should be unmistakably clear.

KEYWORDS

code quality, mining software repositories, business impact, developer productivity, technical debt, software defects

ACM Reference Format:
Adam Tornhill and Markus Borg. 2022. Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Efficient software development is a competitive advantage that allows companies to maintain a short time-to-market [20]. To succeed, companies need to invest in their software to ensure efficient and fast marketplace results [37]. Adding to that challenge, software companies are also facing a challenge in that there is a global shortage of software developers [13]; demand substantially out-weights supply. Moreover, several analyses forecast that the shortage of software developers will only be exacerbated as the digitalization of society continues [11, 13, 29, 43, 48]. At the same time as the software industry is struggling with recruiting enough talent, research indicates that up to 42% of developers' time is wasted dealing with Technical Debt (TD) where the cost of subpar code alone comes to $85 billion annually [57]. This implies that there is an untapped potential in software projects if the code quality is improved and TD paid down.

Unfortunately, the software industry often moves in the opposite direction: research finds that developers are frequently forced to introduce new TD [10] as companies keep trading code quality for time to market and new features [24]. This is a decision-making bias known as *hyperbolic discounting*, i.e., humans make choices today that their future selves would prefer not to have made [35].

One reason for the hyperbolic discounting of code quality is that the business impact of code quality remains vague [32]. This was painfully visible in a study of 15 large software organizations where the benefits of paying down TD is not always clear to managers, and consequently some managers would not grant the necessary budget, nor priorities, for refactoring [38]. The lack of clear and quantifiable benefits makes it hard to build a business case for code quality and, hence, easy to trade short-term wins for long-term sustainability and software maintenance; there are no standard Key Performance Indicators (KPIs) for code quality that are relevant to companies in the way that financial KPIs are [5]. Consequently, in a study involving 1,831 participants, only 10% reported that their business manager was actively managing TD [21]. To make a change, we conclude that TD needs visibility throughout the whole organization, which includes business managers, not just developers and architects.

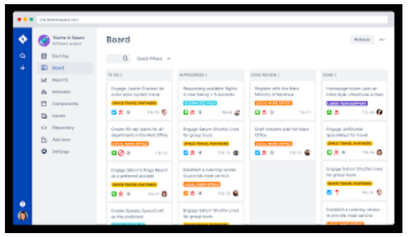
Second, software organizations lack a way of tracking the time wasted on TD with sufficient accuracy [26]. Further, enforcing such detailed time tracking could be perceived as exercising too much control over employees [51] as well as adding too much administrative burden [10]. In consequence, Besker *et al.* report from a study surveying 43 developers that "none of the interviewed companies had a clear strategy on how to track and address the wasted time" [10]. This means that while the overall development costs (e.g. staffing) are known, there is a need for better ways of mapping those overall costs to the additional time spent working on production code of various quality, i.e., the TD interest.

In this paper, we report how code quality impacts development time in 39 proprietary production codebases. We do that by considering CodeScene's Code Health metric[1] as a proxy for code quality. Moreover, we explore an algorithm to capture the development

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

[1] Research publication: https://arxiv.org/abs/2203.04374

Time-In-Development: how do we measure it?

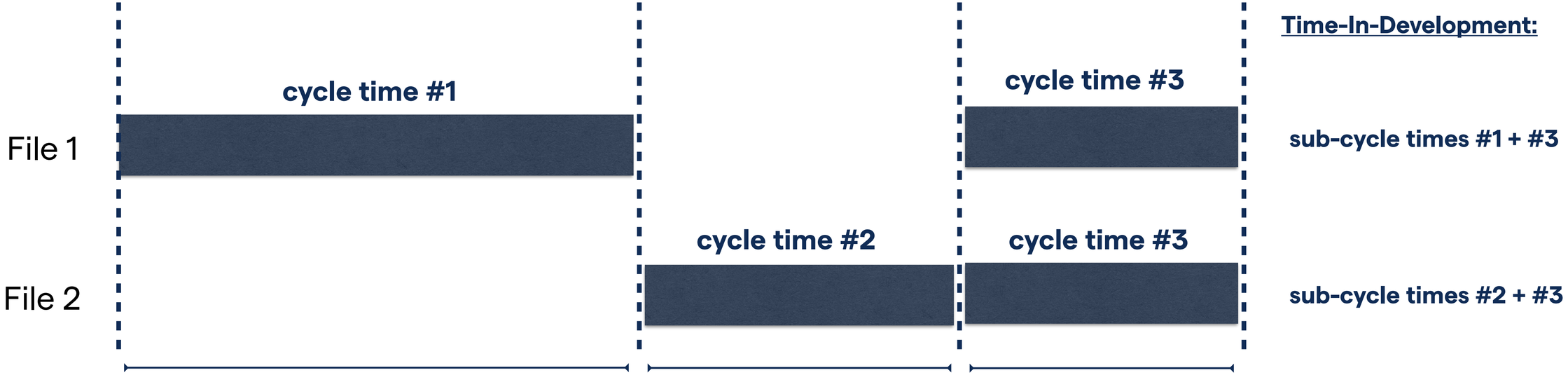


Data source: Jira

Data source: Jira + Git

Jira Issue X moved to “In Progress”:
starts the sub-cycle time #1

commit #1 commit #2 commit #N

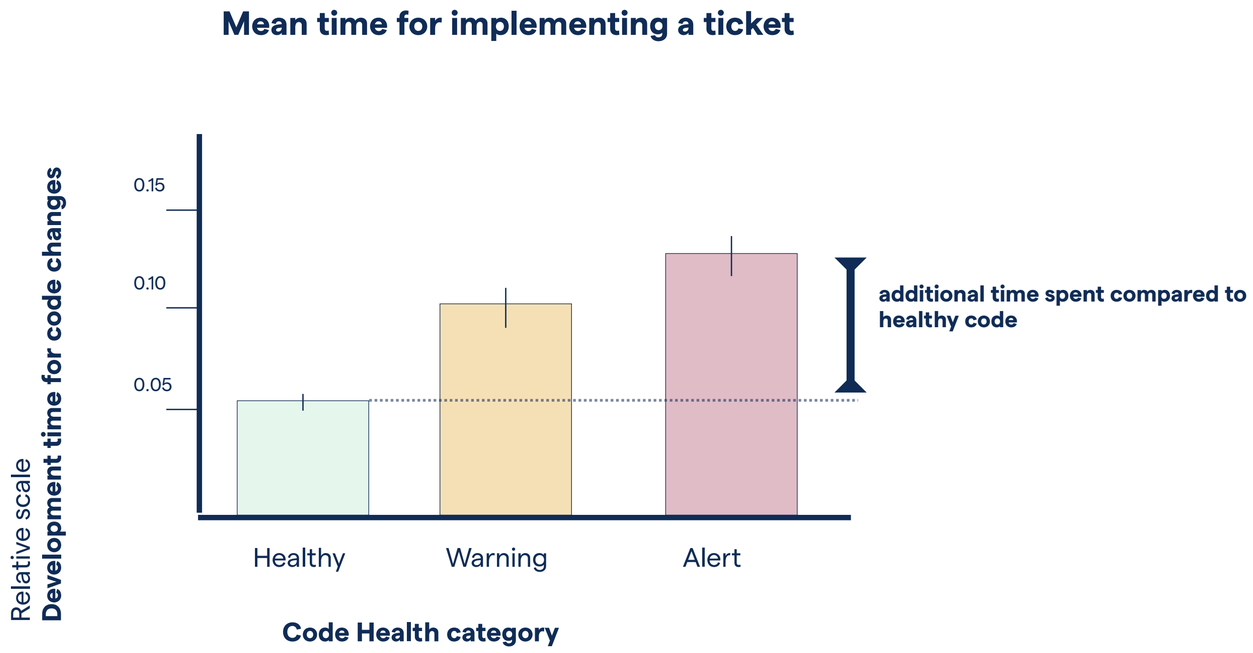


Time-In-Development:

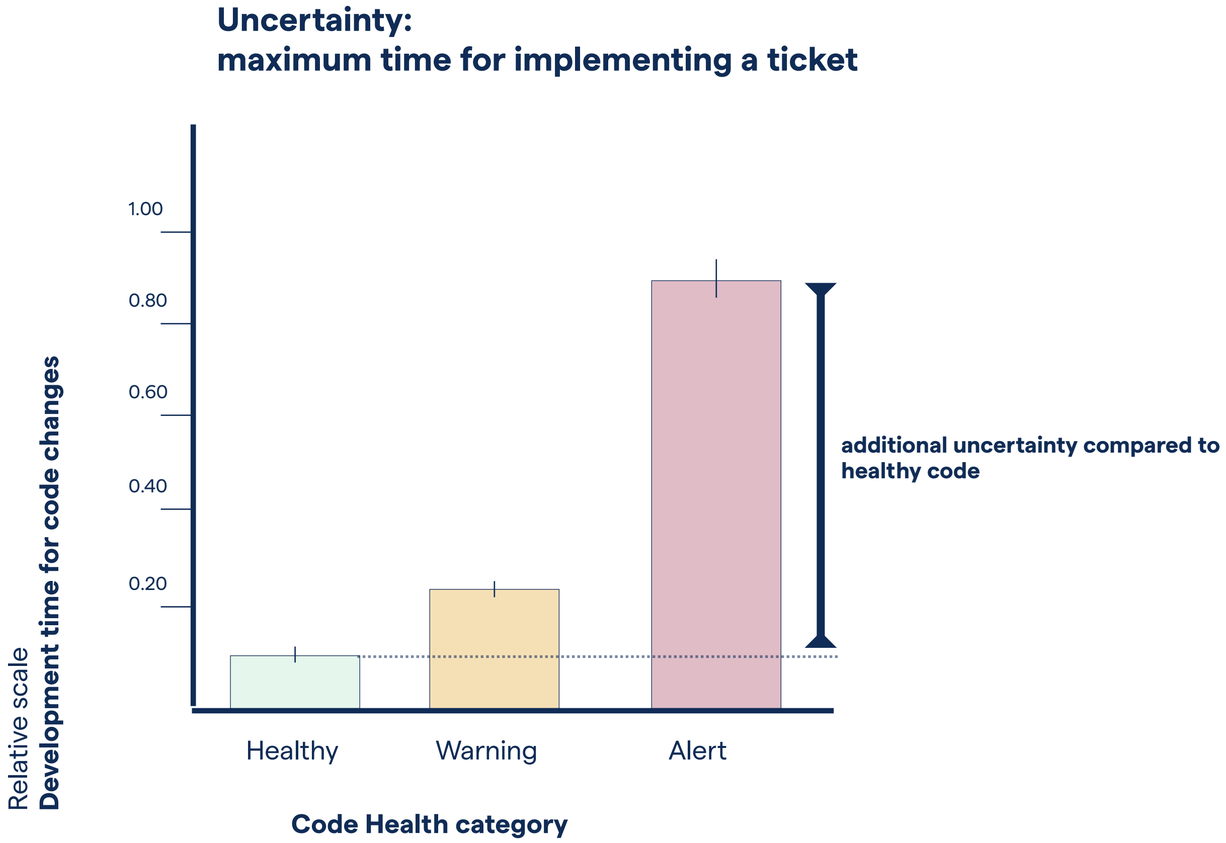
The results:

Does code quality matter?

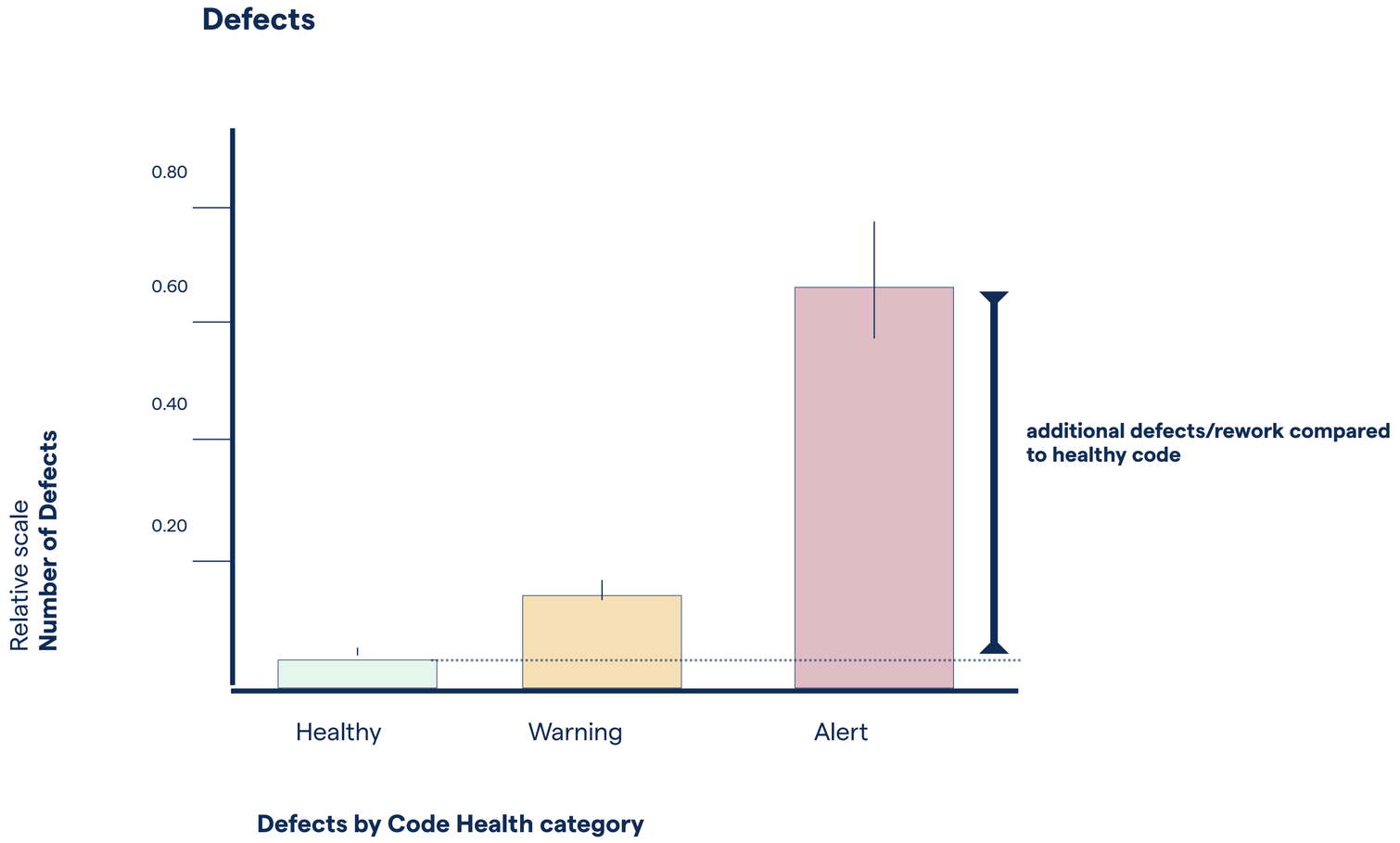
Green Code: Implementing a feature is twice as fast



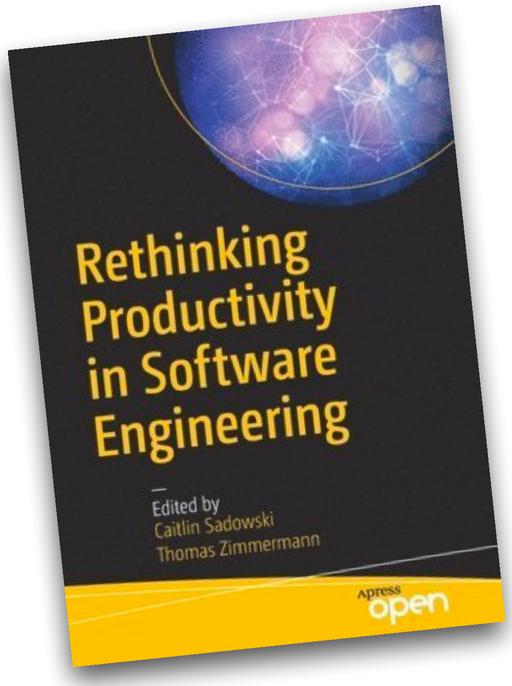
Red Code: A feature can take up to 9 times longer



Red Code: contains 15 times more defects



The programmer perspective: **how code quality impacts development teams**



The most frequent causes of unhappiness:

1. Stuck in problem-solving
2. Time pressure
3. Work with bad code

“[Developers] suffer tremendously when they meet bad code that could have been avoided in the first place”

Grazitotin, D., & Fagerholm, F. (2019). “Happiness and the Productivity of Software Engineers"

Theory into practice: how would we use this data?

Code quality constraints a business

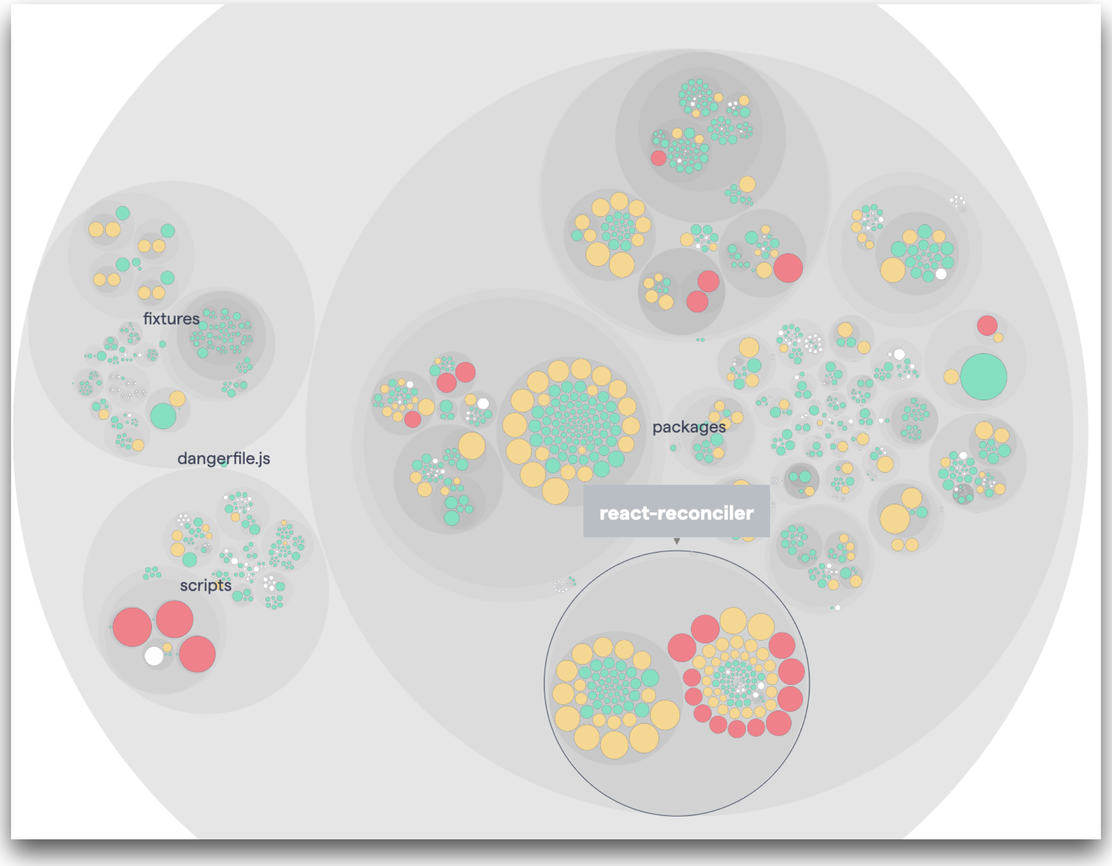
- Give all stakeholders — devs, product, management — the same situational awareness of where the strong and weak parts are.

Fight hyperbolic discounting:

- Discussing future risks primes you for starting to address them.

Build a business case for improvements:

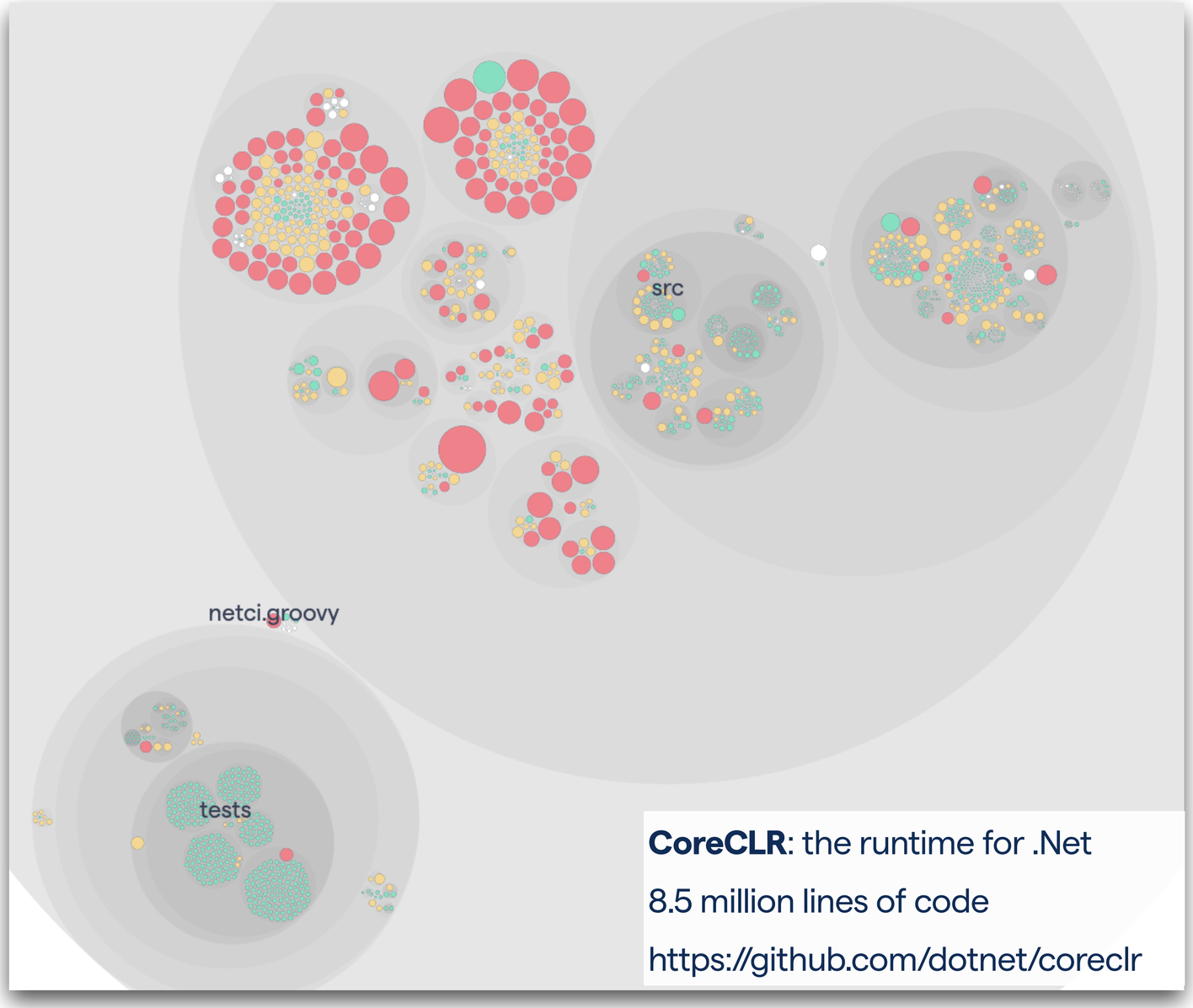
- Refactoring and larger improvements can come with a business expectation.



Making it actionable by combining people +code:

Prioritize remediation to large amounts of Red Code

Red Code: Where do we start?

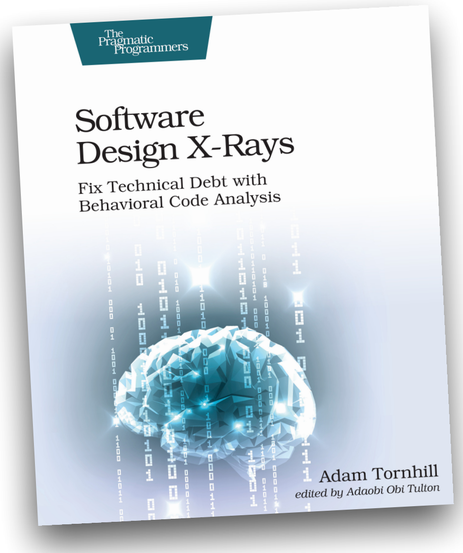


CoreCLR: the runtime for .Net

8.5 million lines of code

https://github.com/dotnet/coreclr

What is a **Behavioral Code Analysis?**



Behavioral Code Analysis = code + people + context

While the code is important, it's even more important to understand how we — as a development organisation — interact with the system we're building.

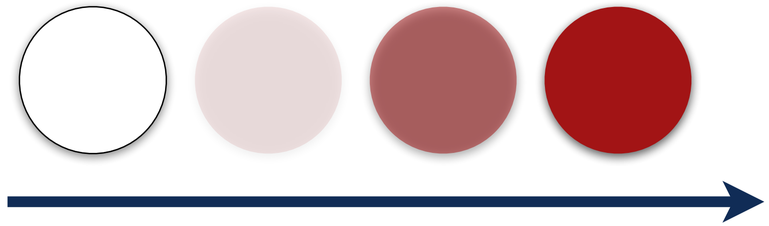
Version-Control — **A Behavioral Data Source**

```
Commit: b557ca5
Date: 2016-02-12
Author: Kevin Flynn

    Fix behavior of StartsWithPrefix

8    27   src/Mvc.Abstractions/ModelBinding/ModelStateDictionary.cs
1    10   src/Mvc.Core/ControllerBase.cs
1    1    src/Mvc.Core/Internal/ElementalValueProvider.cs
1    39   src/Mvc.Core/Internal/PrefixContainer.cs

Commit: fd6d28d
Date 2016-02-10
Author: Professor Falken

   Make AddController not overwrite existing IControllerTypeProvider

8    1    src/Core/Internal/ControllersAsServices.cs
48   0    test/Core.Test/Internal/ControllerAsServicesTest.cs
13   0    test/Mvc.FunctionalTests/ControllerFromServicesTests.cs

Commit: 910f013
Date :2016-02-05
Author Lisbeth Salander

   Fixes #4050: Throw an exception when media types are empty.

20   1    src/Mvc.Core/Formatters/InputFormatter.cs
```

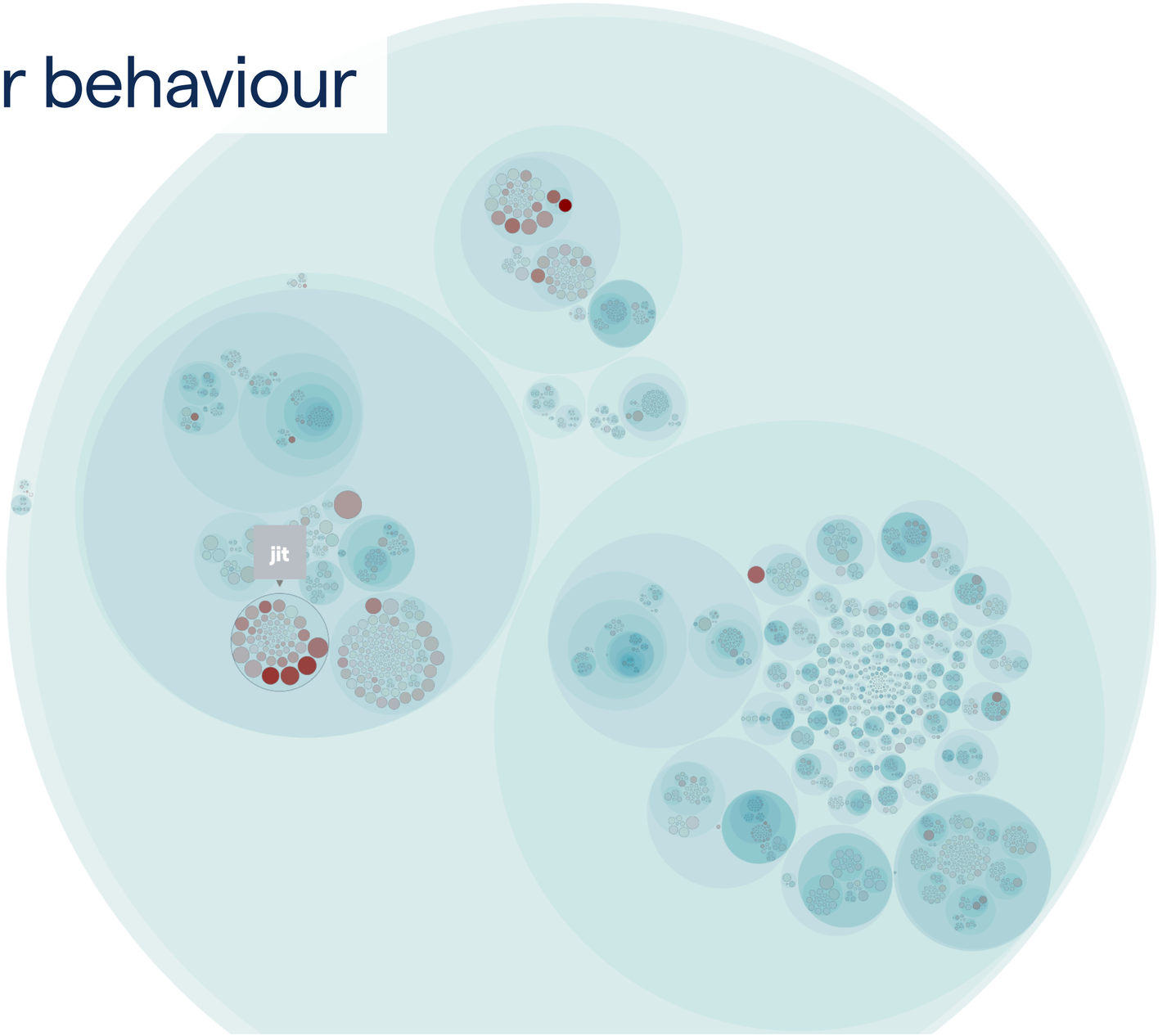
A Time Dimension

Hotspots:

Prioritize based on developer behaviour

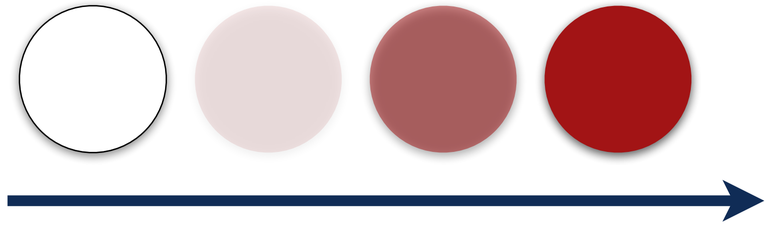


Interest rate: Code Change Frequency



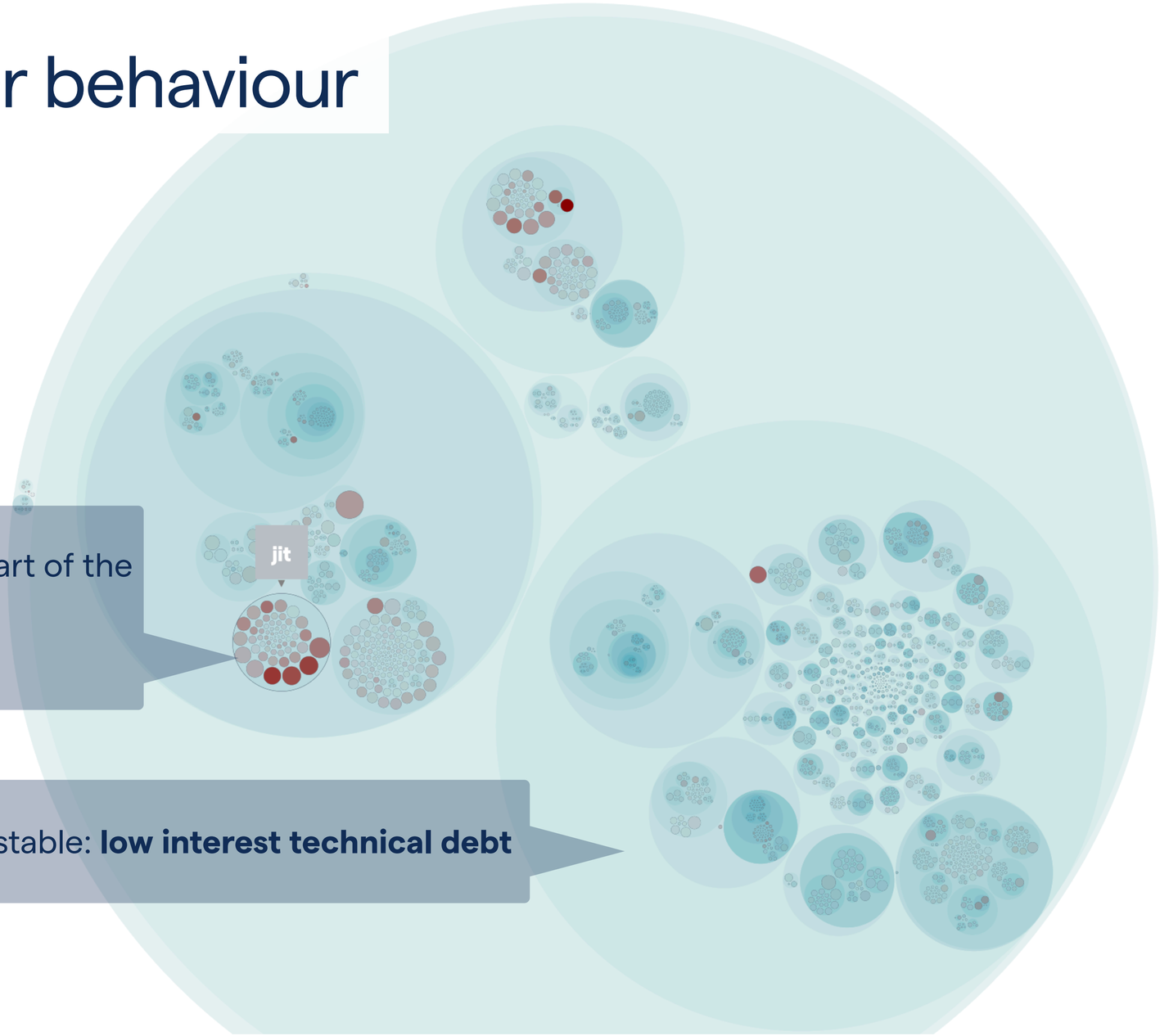
Hotspots:

Prioritize based on developer behaviour



Interest rate: Code Change Frequency

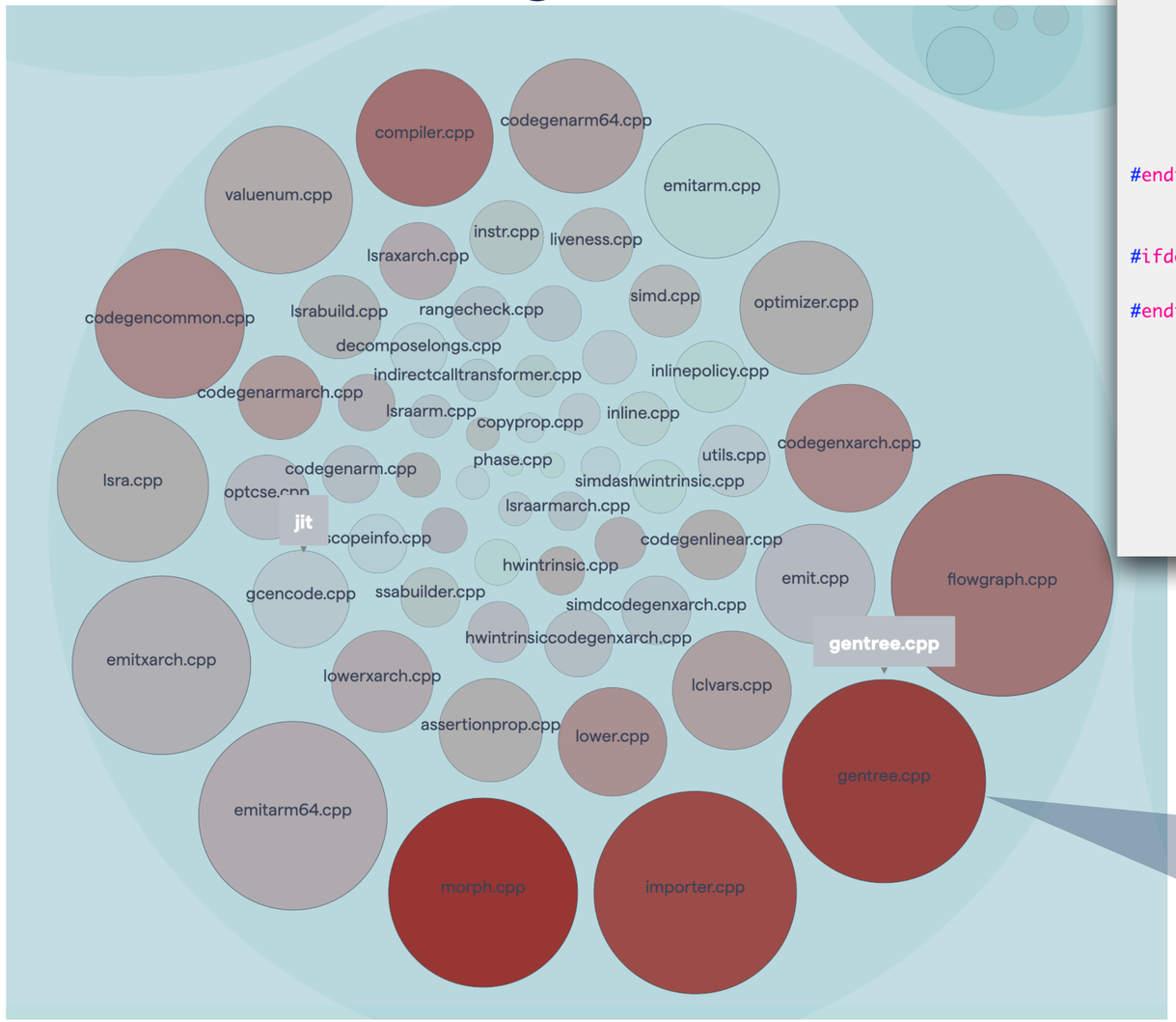
Most development activity is in a small part of the codebase: **high interest technical debt**



Most code is stable: **low interest technical debt**

CodeScene

A look into the *Jit* package:
Actionable Insights?

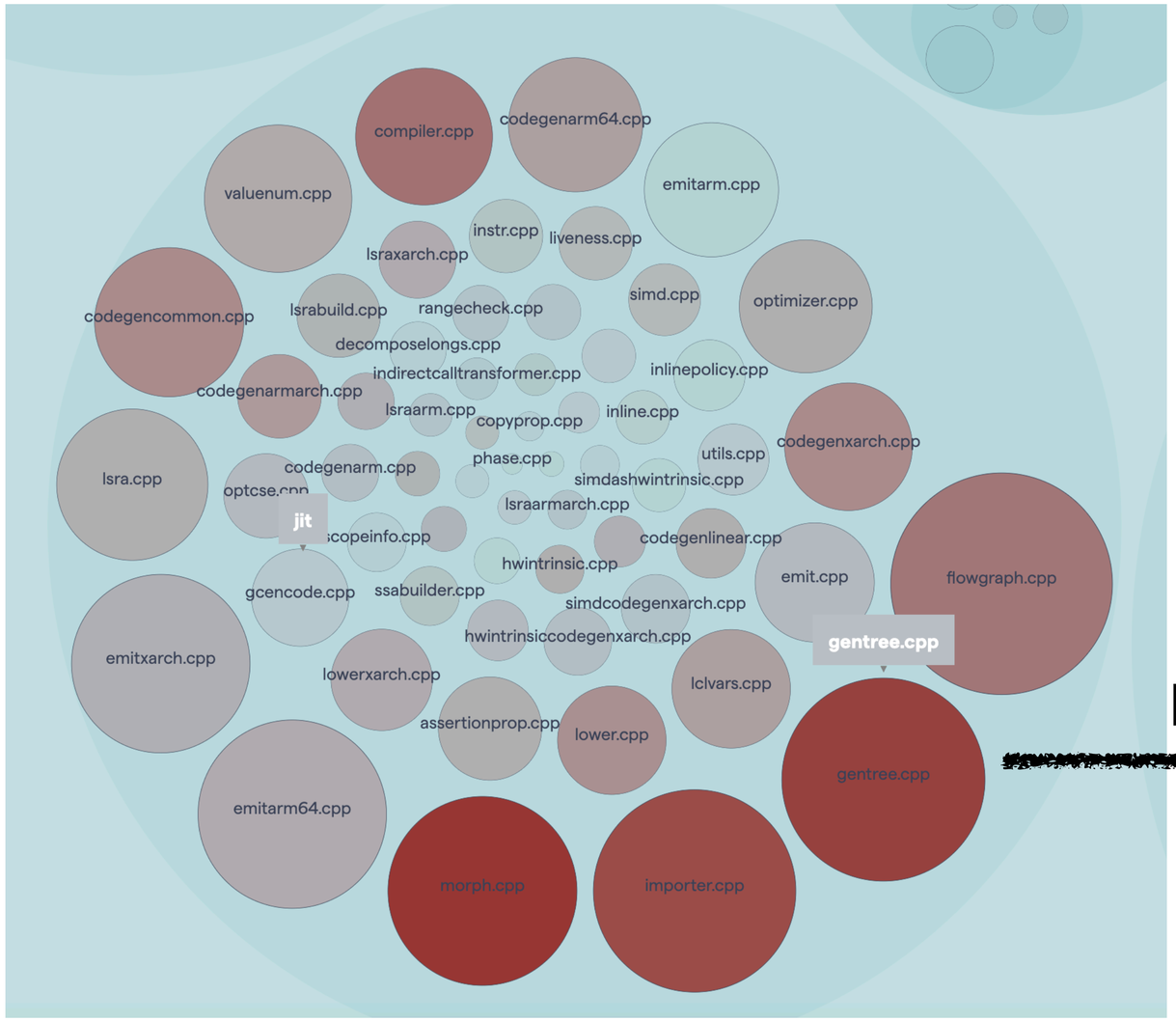



```
    if (kind & (GTK_CONST | GTK_LEAF))
    {
        switch (oper)
        {
            case GT_CNS_INT:

#if defined(LATE_DISASM)
                if (tree->IsIconHandle())
                {
                    copy =
                        gtNewIconHandleNode(tree->AsIntCon()->gtIconVal, tree->gtFlags, tree->AsIntCon(
                    copy->AsIntCon()->gtCompileTimeHandle = tree->AsIntCon()->gtCompileTimeHandle;
                    copy->gtType                          = tree->gtType;
                }
                else
#endif
                {
                    copy = gtNewIconNode(tree->AsIntCon()->gtIconVal, tree->gtType);
#ifdef DEBUG
                    copy->A
#endif
                    copy->A
                    copy->A
                }
                goto DONE;

            case GT_CNS_LNG
                copy = gtNe
                goto DONE;
```

```
case GT_LCL_VAR:

    if (tree->AsLclVarCommon()->GetLclNum() == varNum)
    {
        copy = gtNewIconNode(varVal, tree->gtType);
        if (tree->gtFlags & GTF_VAR_ARR_INDEX)
        {
            copy->LabelIndex(this);
        }
    }
    else
    {
        // Remember that the LclVar node has been cloned. The flag will
        // be set on 'copy' as well.
        tree->gtFlags |= GTF_VAR_CLONED;
        copy = gtNewLclvNode(tree->AsLclVar()->GetLclNum(),
                             tree->gtType DEBUGARG(tree->AsLclVar()->gtLclILoffs));
        copy->AsLclVarCommon()->SetSsaNum(tree->AsLclVarCommon()->GetSsaNum());
```

14,000 Lines of Code!

Hotspots: X-Ray: `gentree.cpp`



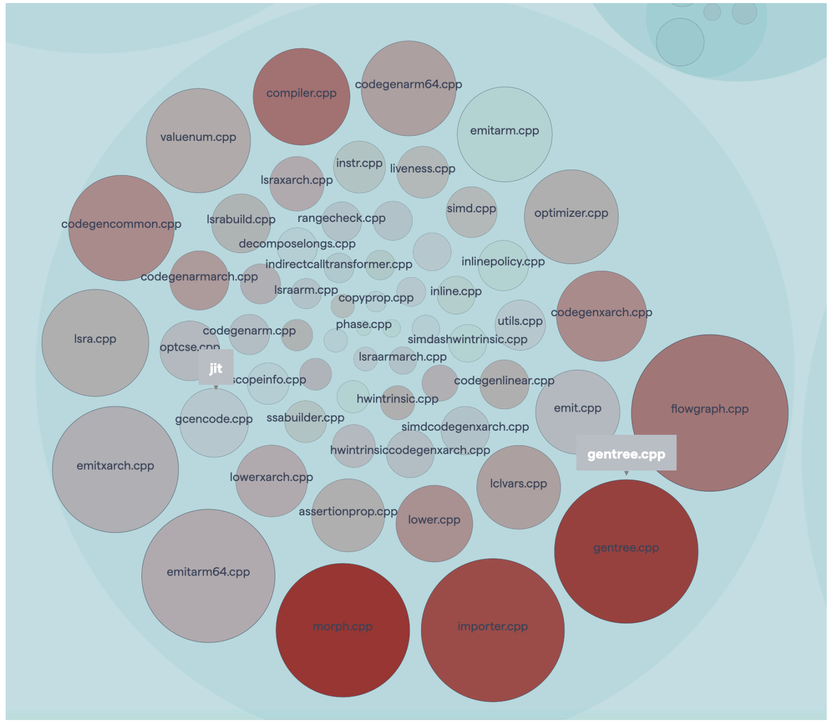
Parse

Function Level Hotspots

Recommended functions to improve.

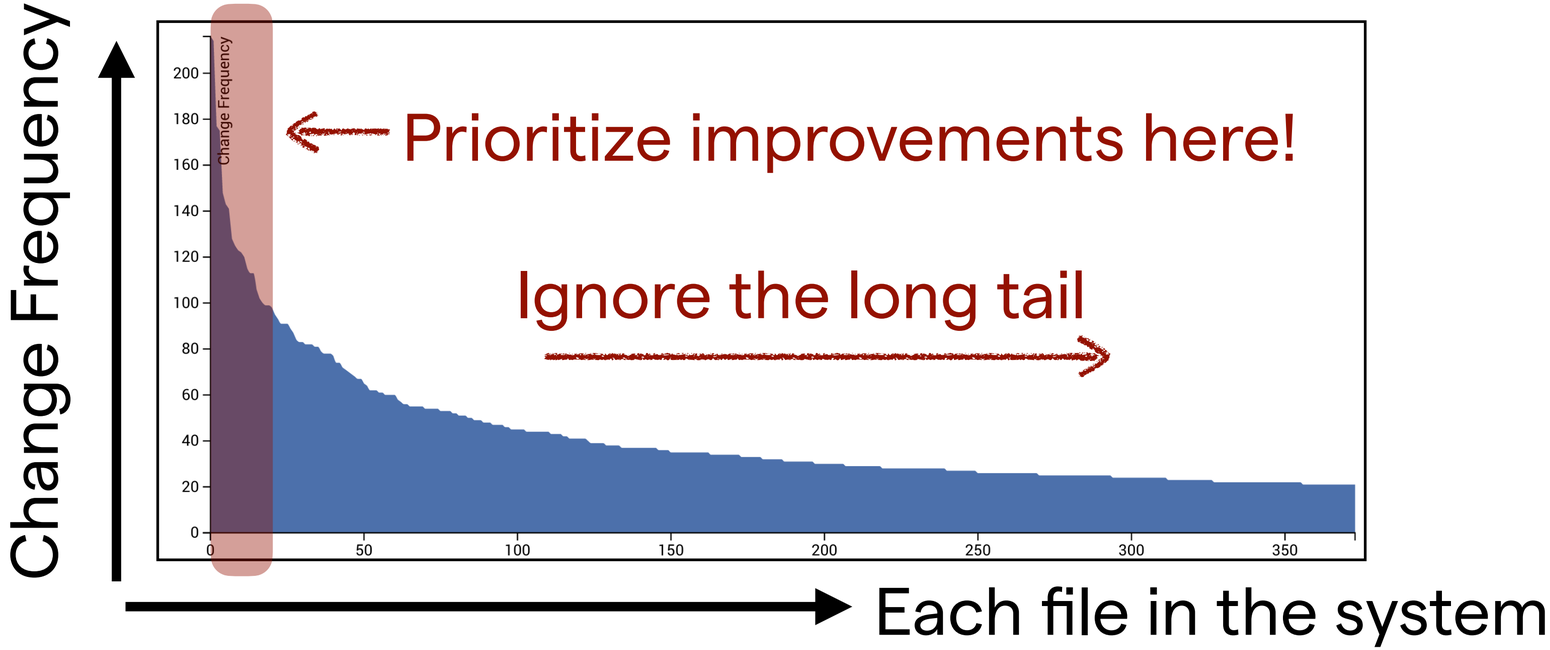
From https://pragprog.com/book/atevol/software-design-x-rays

X-Ray of `gentree.cpp`



Function	Change Frequency	Lines of Code	Cyclomatic Complexity
`Compiler::gtCloneExpr` View Complexity Trend | View Function Code	42	586	112
`Compiler::gtHashValue` View Complexity Trend | View Function Code	40	335	66
`GenTree::Compare` View Complexity Trend | View Function Code	34	390	110
`Compiler::gtSetEvalOrder` View Complexity Trend | View Function Code	29	1,531	291
`Compiler::gtDispTree` View Complexity Trend | View Function Code	23	578	130
`Compiler::gtDispNode` View Complexity Trend | View Function Code	18	510	125

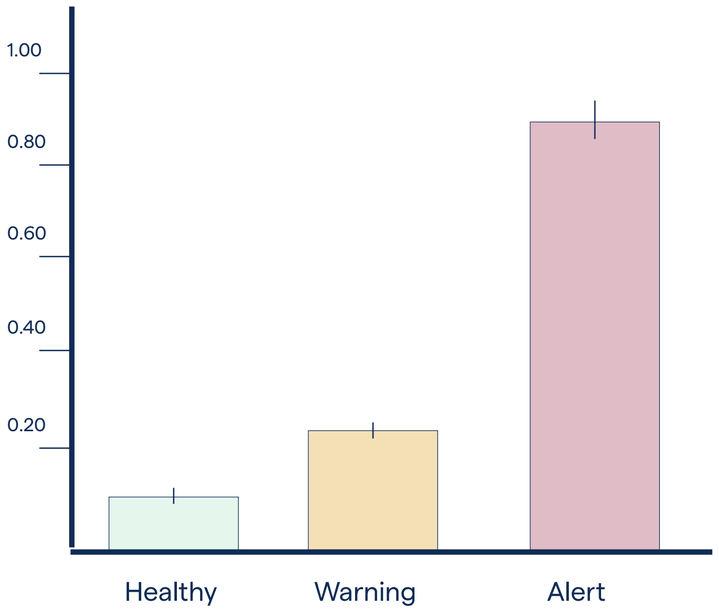
Hotspots: why you don't have to fix all tech debt



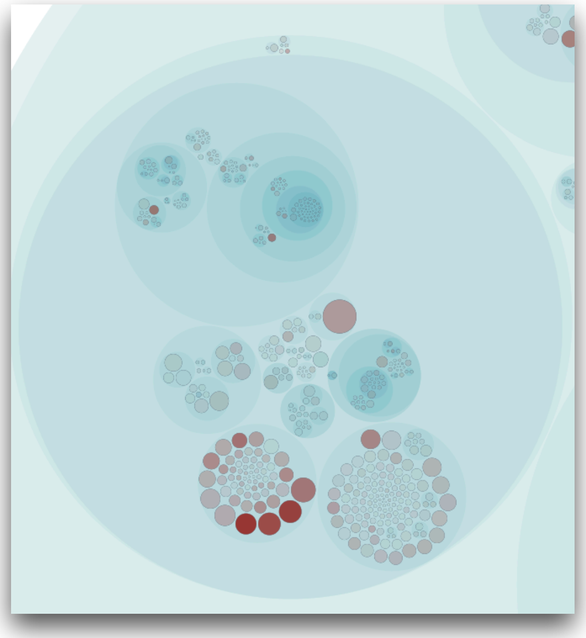
Key take-aways:

- Most code is in the long-tail. This is low-interest debt.
- Hotspots only make up 2-4% of the total codebase, but attract 20-70% of all development activity!
- Code health issues in a hotspot are expensive. This is high-interest debt.

Technical debt: a data-driven approach



Quality dimension: Code Health identifies risks and opportunities.

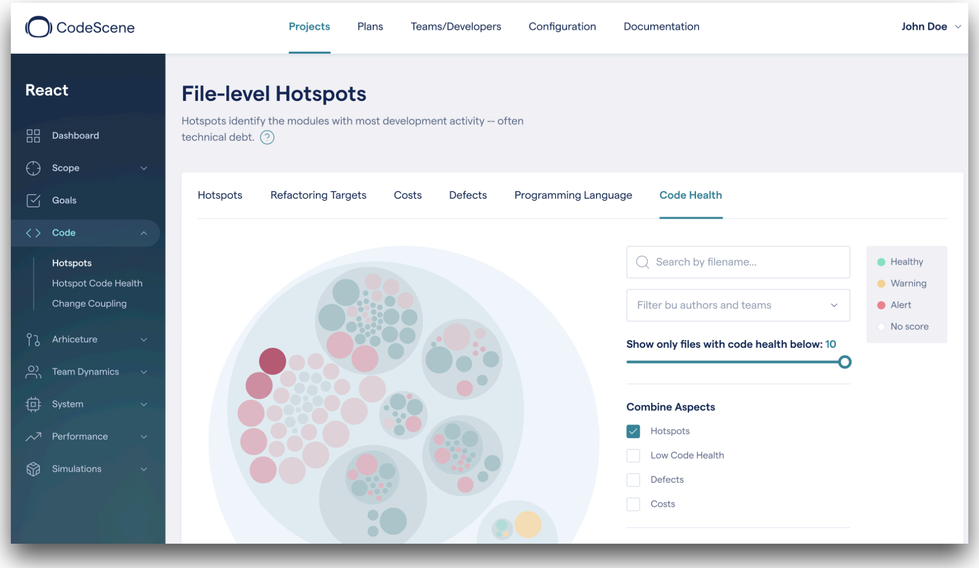


Hotspot dimension: what's the impact and priorities?

Speed + Quality: you can have it all

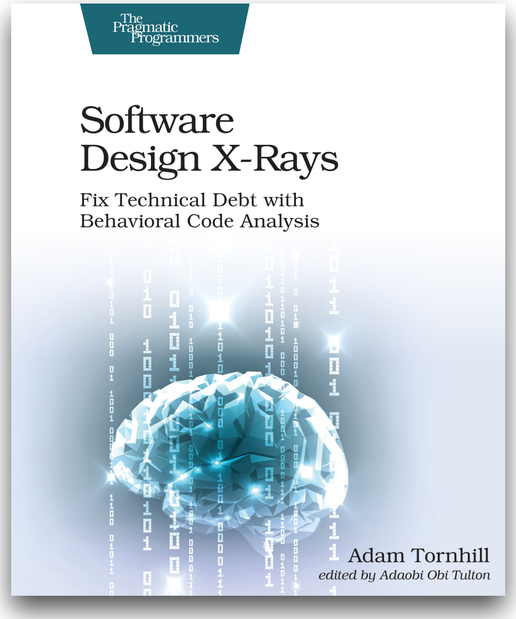
> “Our results indicate that improving code quality could free existing capacity; with 15 times fewer bugs, twice the development speed, and 9 times lower uncertainty in completion time, the business advantage of code quality should be unmistakably clear.”
>
> A. Tornhill & M. Borg (2022)

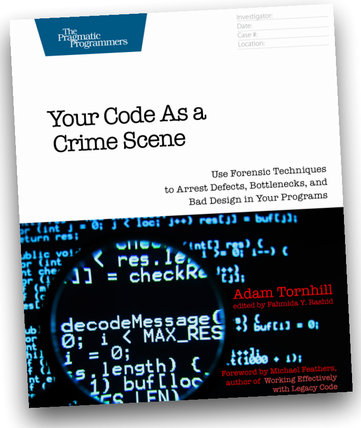
Code health tool: https://codescene.com/



The Code Red papers:

- White paper: https://codescene.com/hubfs/web_docs/Business-impact-of-low-code-quality.pdf
- Research publication: https://arxiv.org/abs/2203.04374





Adam Tornhill

https://twitter.com/AdamTornhill

https://se.linkedin.com/company/codescene