yonder

R&D Metrics, Reloaded

Portfolio Managers Focus Group

2025 May, Manoir St Sauveur Qc

agenda

1 **Introduction**
What is technical risk and technical debt

2 **Measuring**
How is software quality measured

3 **Impact**
How do we calculate the impact?

yonder

Remus Pereni

CTO / Software Architect
Technical University of Cluj Napoca
BS, Computer Science

- **1996 – Nethrom (Yonder)**
- **2000 – Startup**
- **2005 – Yonder / SD, PM, DM, Architect**
- **2017 – CTO**

2016 start of the Technology DDs
200+ Technology DD Reports
TSS Blue, TSS Public, Vela, Harris, Perseus, CSI, Strikwerda Investments, Jonas, Volaris

The impact of code and technical debt

1 Introduction

yonder

Technical Risk

The potential for losses due to failures or shortcomings in technology systems, processes, or implementations that can impact project outcomes or business objectives.

Is that different from **Technical Debt**?

TECHNICAL RISK

Technical Debt

The accumulated costs and future liabilities resulting from shortcuts or suboptimal technical decisions made during the software development process. It represents the work that needs to be done before a piece of software can be considered complete or optimal.

- Code debt
- Architectural debt
- Technology debt
- Testing debt
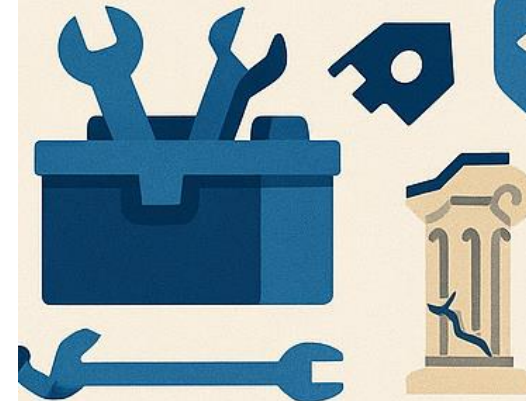- Infrastructural debt
- Know-how debt

Technical Debt Quadrants

DELIBERATE

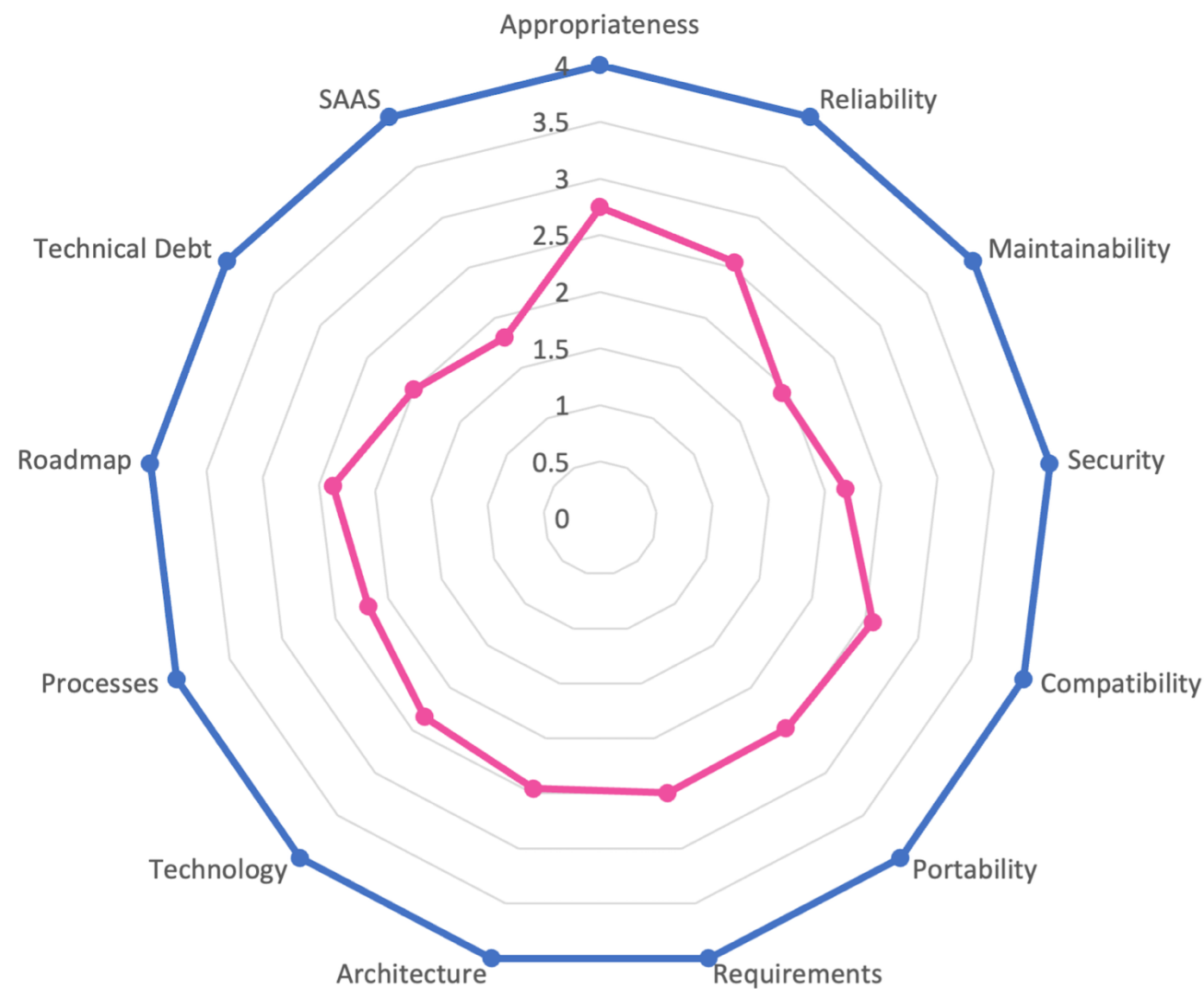|	
" We don't have time for design"	*" We must ship now and deal with the consequences"*
"What's Layering?"	*" Now we know how we should have done it"*

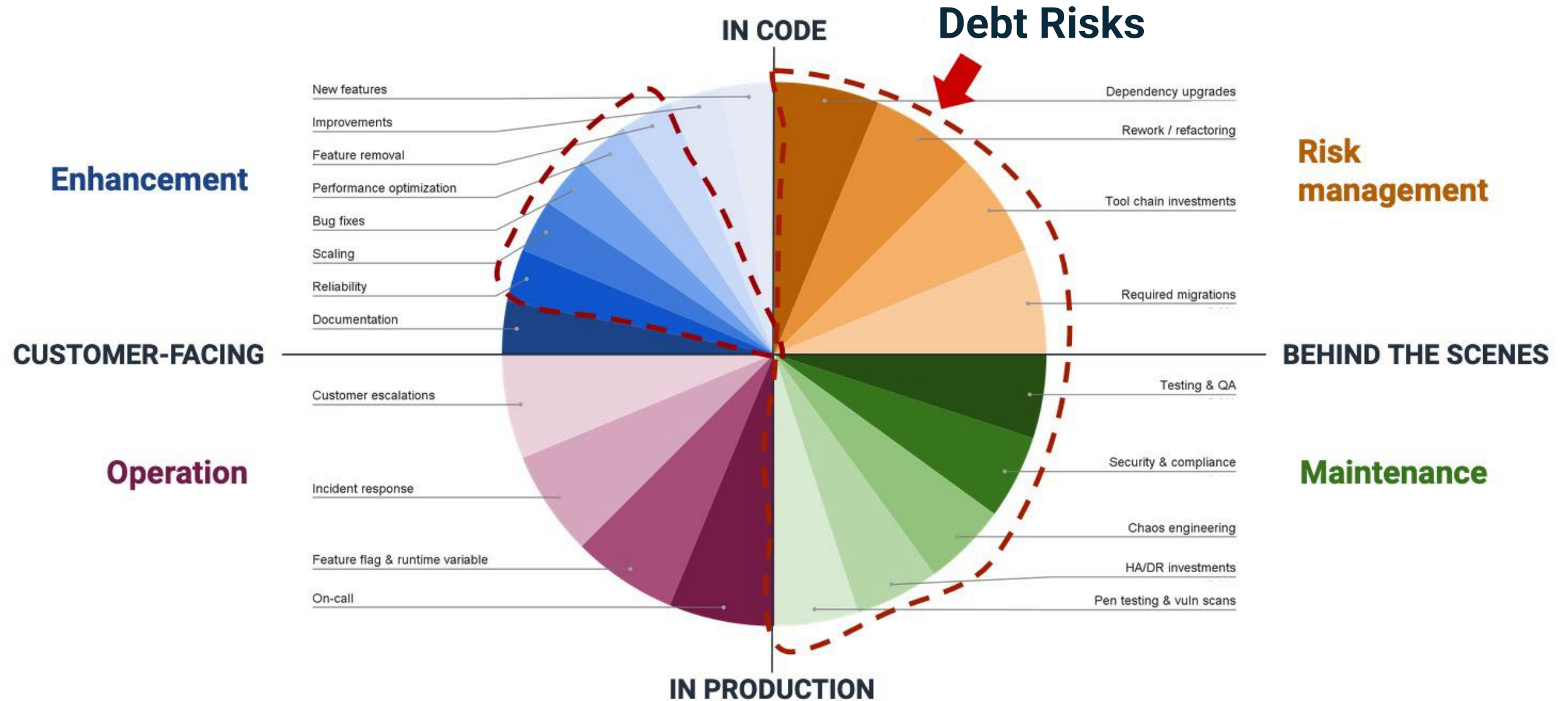**RECKLESS** ——————————————— **PRUDENT**

ACCIDENTAL

yonder
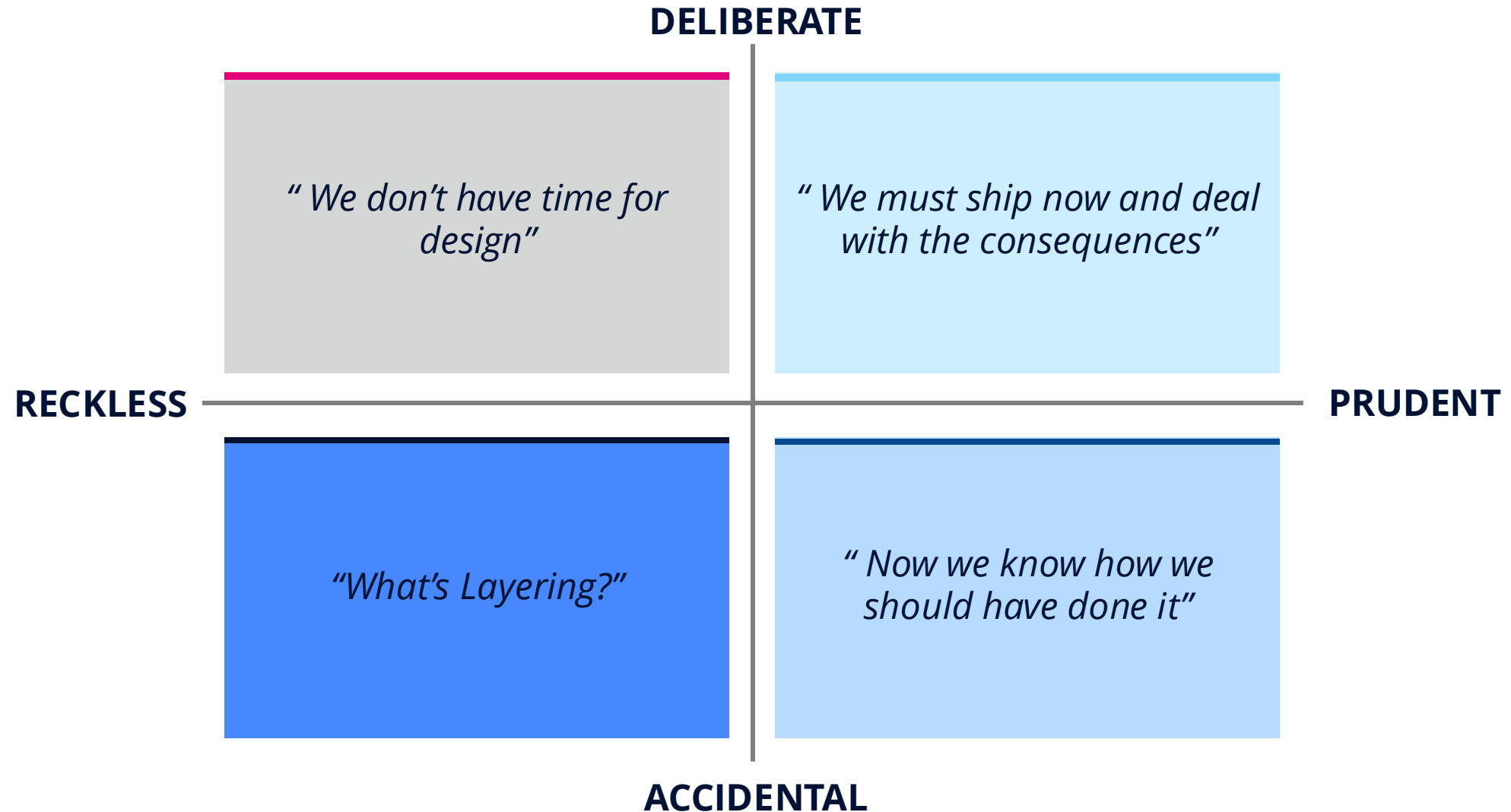
Does your portfolio have a technical debt problem?

yonder

Do we see a tech debt problem?

Avg of 144 products
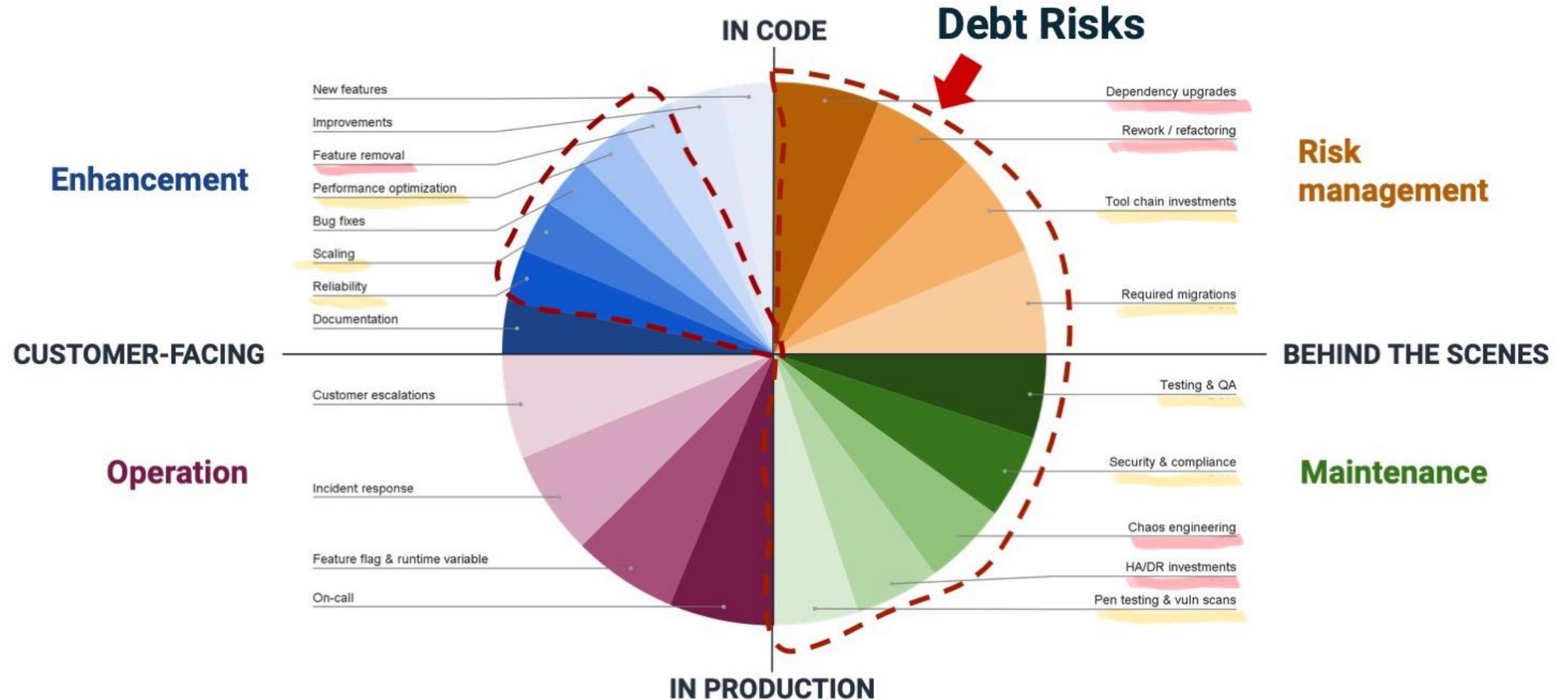seen in the past years
on M&A

yonder

Technical Debt Quadrants

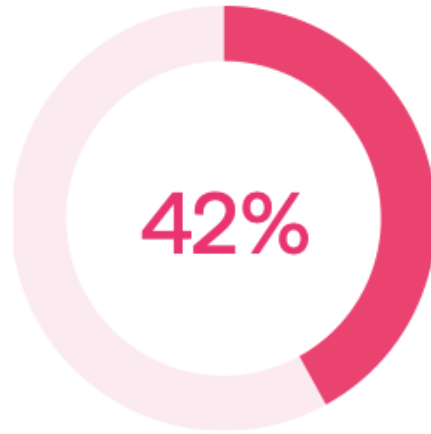Is there **focus** the **risk mitigation**? **Which items**?

Technical Debt Quadrants

What is your process to manage technical debt in your portfolio?

yonder

Who measures technical debt?

10%

Only 10% of business managers actively manage technical debt.

42%

Developers waste 42% of the work week on technical debt.

none of the interviewed companies had a clear strategy on how to track and address the wasted time

Besker, T., Martini, A., Bosch, J. (2019) "Software Developer Productivity Loss Due to Technical Debt"

Antonio Martini, Terese Besker, and Jan Bosch. 2018. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Science of Computer Programming 163 (2018), 42–61.

yonder

TSS Security Control Framework a.k.a. TSS SCF

In TSS the TSS SCF was introduced in 2022 and covers areas that are related to security:

- **penetration tests,**
- **data encryption**
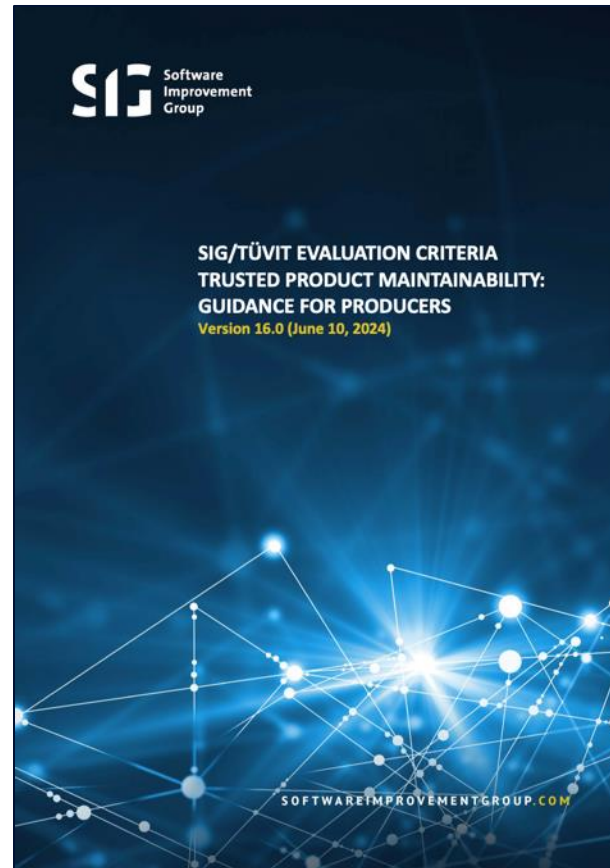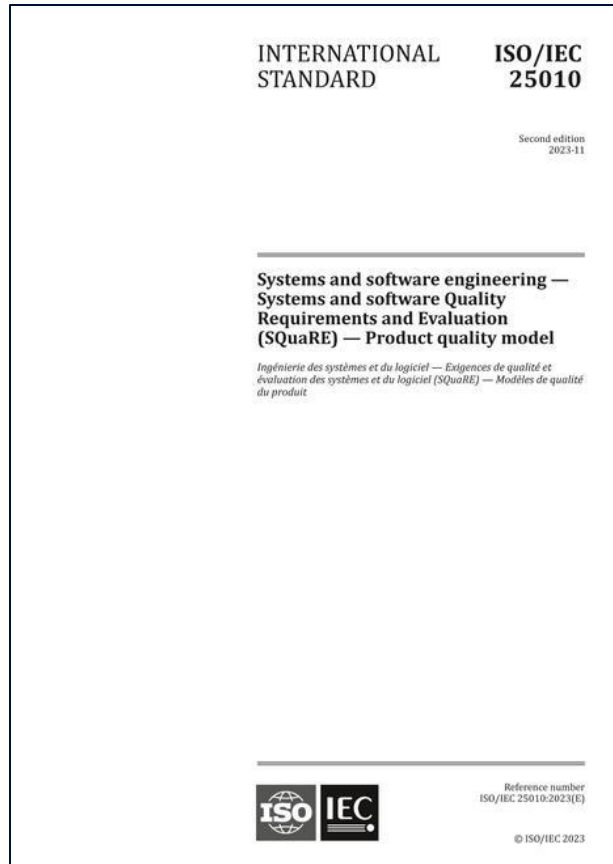- **security requirements (OWASP ASVS)**
- **unsupported software**

Impact is high, companies see vulnerable or unsupported third-party dependencies as uncompliant and leads to pressure to change including in M&A.

Until now MD had to sign that he acknowledges the risks, and fines can be applied for not addressing the issues.

logoipsum   yonder

2 Measuring

Measuring software quality

https://www.softwareimprovementgroup.com/wp-content/uploads/SIG-TUViT-Evaluation-Criteria-Trusted-Product-Maintainability-Guidance-for-producers.pdf

https://www.iso.org/standard/80623.html   https://www.iso.org/standard/78176.html

yonder

ISO / IEC 25010:2011 Product Quality Model

Overall Rating: ★★☆☆☆

Functionality	Reliability	Efficiency	Usability	Maintainability	Security	Compatibility	Portability
Completeness	Maturity	Time behavior	Recognizability	Modularity	Confidentiality	Co-existence	Adaptability
Correctness	Availability	Resource Utilization	Learnability	Reusability	Integrity	Interoperability	Installability
Appropriateness	Fault Tolerance	Capacity	Error handling	Analyzability	Non-repudiation		Replaceability
	Recoverability		User interface aesthetics	Modifiability	Authenticity		
			Accessibility	Testability	Accountability		
			Operability				

yonder

R&D Metrics

	A	B	C	D	E	F
1		Classification: Confidential		Filled in by:		
2				Reference date:	11 January 2023	
3				Product name:		
4				Product version:		Risk Score	3532
5				Technology	.Net	
6				Total lines of code	154208	
7		R&D Metrics Survey				
8						
9		Question		Answer	Comments	
10						
11	1.0	Code debt				Code Risk Score	232
12	1.1	Percentage of duplicated lines of code from the code base		4,5%		
13		Total duplication risk:		0		
14	1.2	Percentage of code in methods/functions/procedures with more than 15 lines of code		44,0%		
15	1.3	Percentage of code in methods/functions/procedures with more than 30 lines of code		25,8%		
16	1.4	Percentage of code in methods/functions/procedures with more than 60 lines of code		12,6%		
17		Unit size risk score:		32		
18	1.5	Percentage of code in methods/functions/procedures with cyclomatic complexity higher than 5		30,1%		
19	1.6	Percentage of code in methods/functions/procedures with cyclomatic complexity higher than 10		18,7%		
20	1.7	Percentage of code in methods/functions/procedures with cyclomatic complexity higher than 25		7,1%		
21		Total complexity risk:		122		
22	1.8	Percentage of code in methods/functions/procedures with more than 3 argumets		17,2%		
23	1.9	Percentage of code in methods/functions/procedures with more than 5 argumets		6,7%		
24	1.10	Percentage of code in methods/functions/procedures with more than 7 argumets		3,5%		
25		Total method interfacing risk:		78		
26						
27	2.0	Technology debt				Technology Risk Score	3300
28	2.1	Number of dependencies that are unsuported		22		
29	2.2	Number of dependencies that have security vulnerabilities		11		
30	2.2	Number of dependencies that have uncompliant license		0		
31						
32						
33						
34	3.0	Testing quality				
35	3.1	Number of lines of code changed in the last released version		35287	This includes our entire project (.Net and JS)	
36	3.2	Number of bugs that have been reported and accepted as software bugs for the last release		0		
37	3.3	Number of tickets that have been raised for the last release		0		
38		Testing quality risk score		0		
39						Testing Risk Score	0
40						

Code Debt

- **Duplication**
  - % of duplicated lines of code from the codebase should not exceed 4.8%

- % of code in methods / functions / procedures with more than 15/ 30 / 60 lines of code

- % of code in methods / functions / procedures with cyclomatic complexity higher than 5/ 10/ 25

- % of code in methods / functions / procedures with more than 3/ 5/ 7 arguments

Technology Debt

- Number of dependencies that are unsupported

- Number of dependencies that have security vulnerabilities

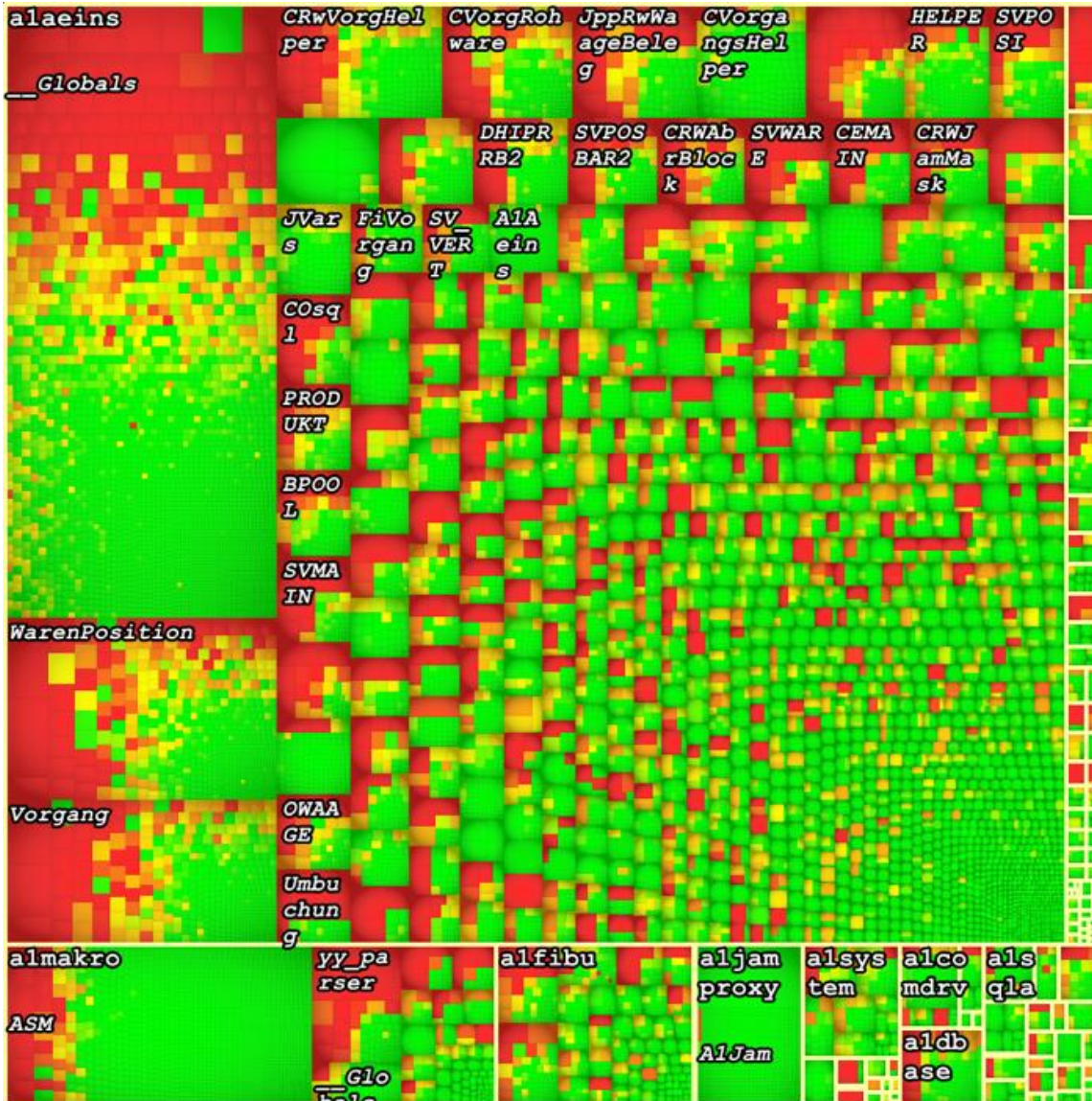- Number of dependencies that have uncompliant license

Testing Quality

- Number of lines of code changed in the last released version

- Number of bugs that have been reported and accepted as software bugs for the last release

- Number of tickets that have been raised for the last release

yonder
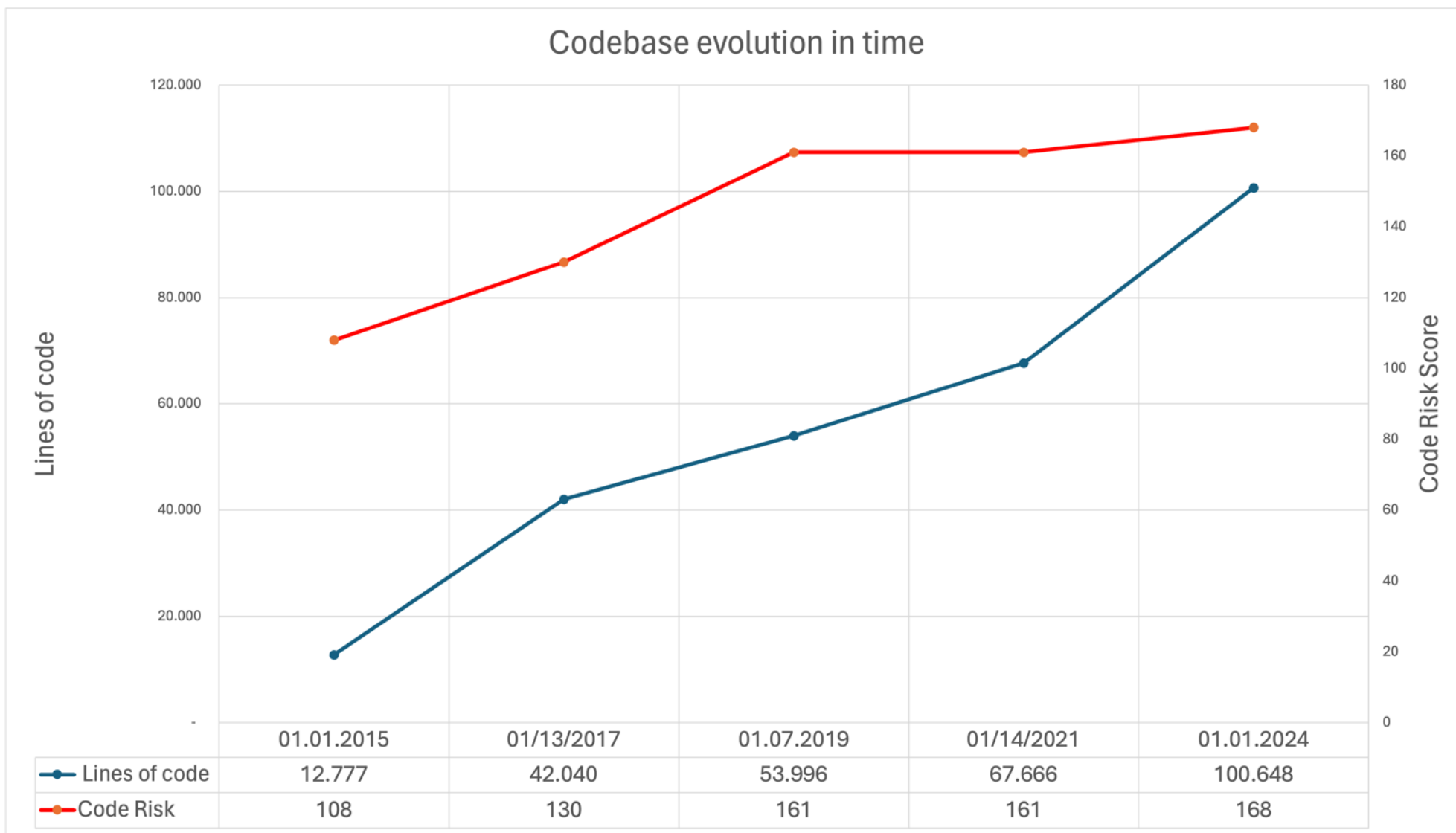
How code debt looks like?
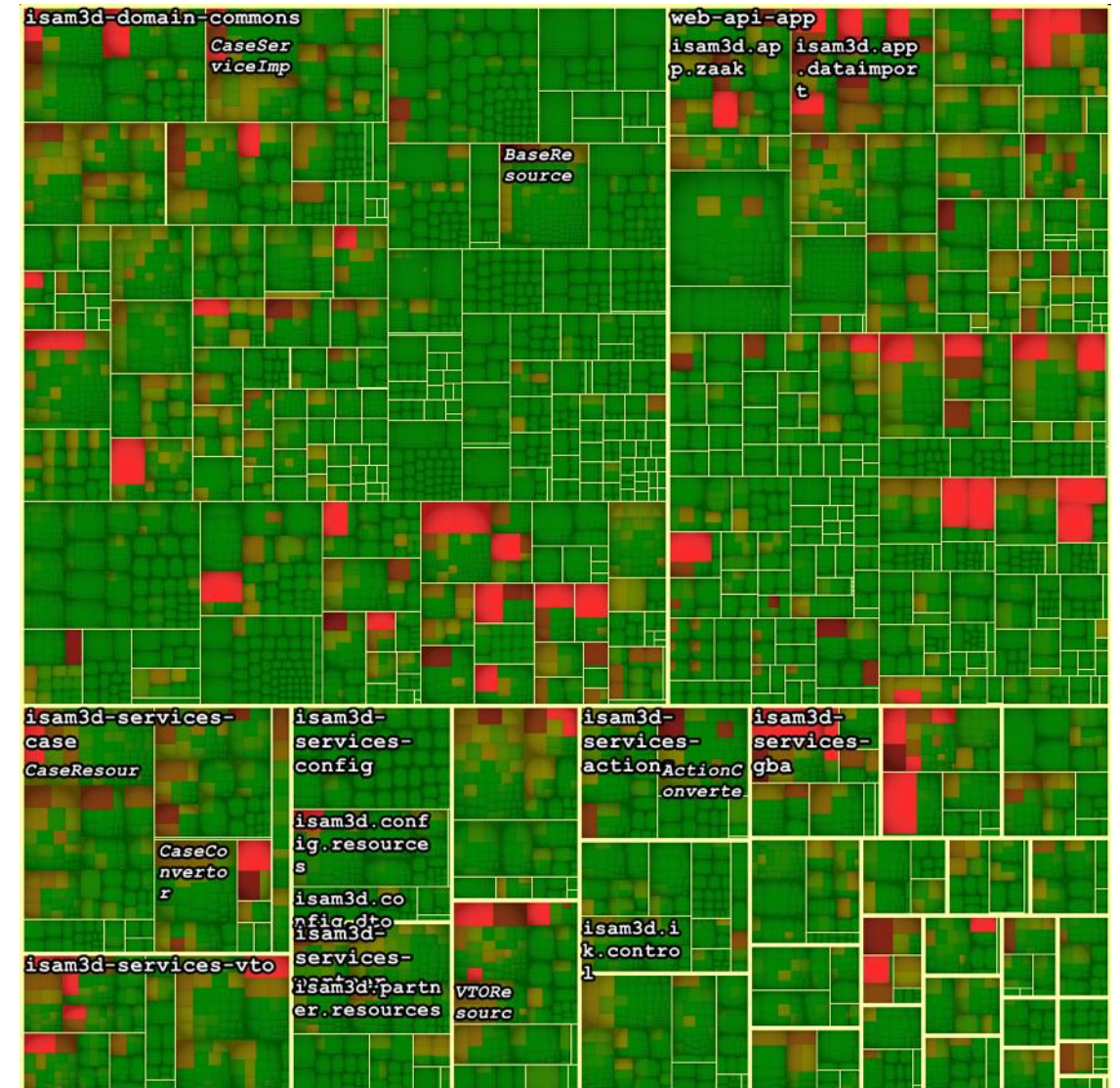

Red: more than 25 decisions / method

Yellow: more than 10 decision / method

Green: more than 5 decision / method

yonder

Looking at a long running project

Codebase evolution in time

	01.01.2015	01/13/2017	01.07.2019	01/14/2021	01.01.2024
Lines of code	12.777	42.040	53.996	67.666	100.648
Code Risk	108	130	161	161	168

yonder

Looking at a long running project

Code debt – why is it important

Codebase evolution in time

	01.01.2015	01/13/2017	01.07.2019	01/14/2021	01.01.2024
Lines of code	12.777	42.040	53.996	67.666	100.648
Jarch. Maint.	83	80	79	78	78

yonder
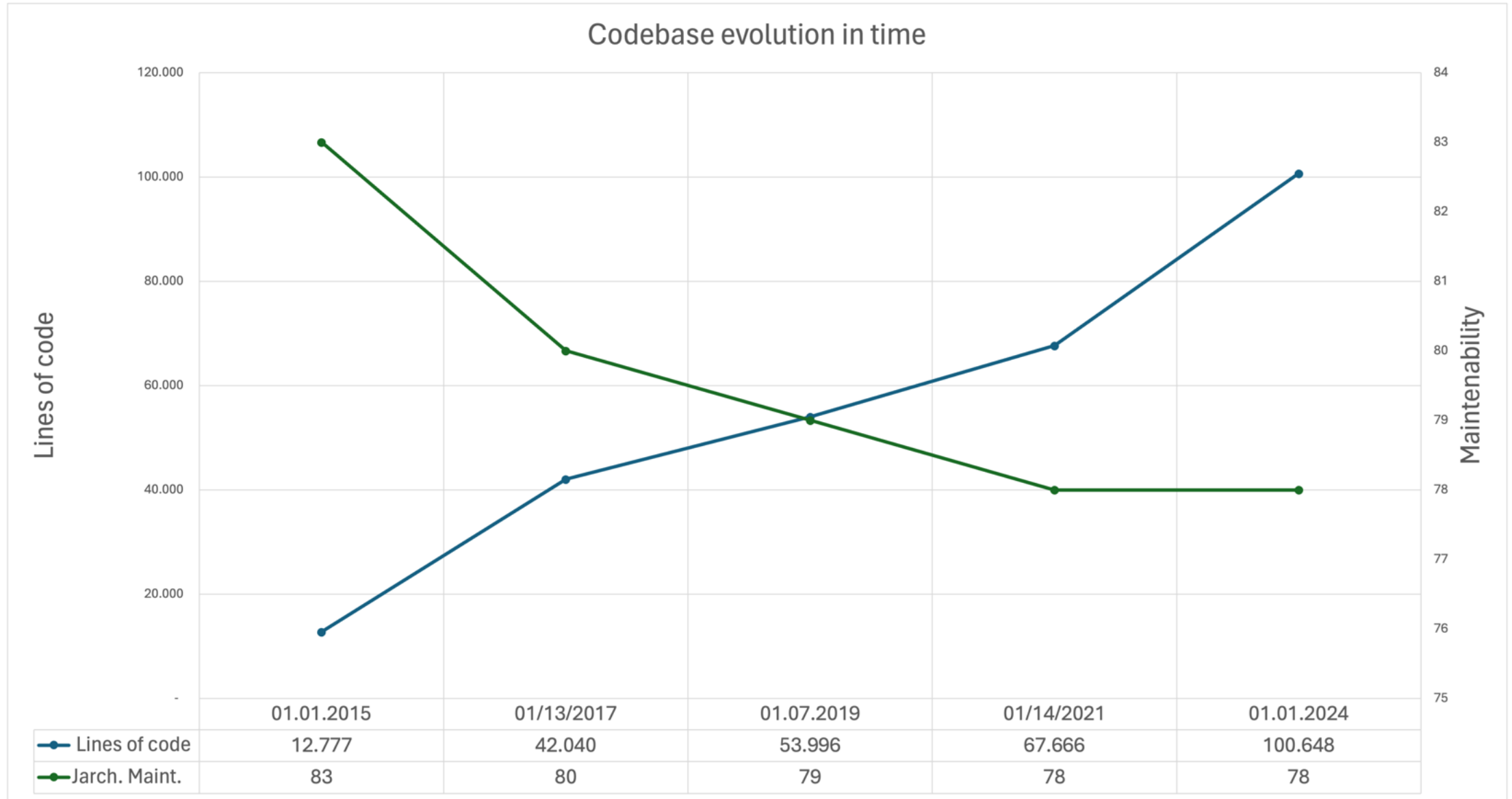
R&D Metrics

24 products measured
2 portfolios
None had zero risk

Product Risk Score

Product Risk Score bar chart (ascending):

- Product 19: 87
- Product 16: 108
- Product 10: 351
- Product 14: 393
- Product 06: 596
- Product 22: 600
- Product 13: 794
- Product 01: 845
- Product 18: 898
- Product 07: 974
- Product 04: 977
- Product 11: 1.097
- Product 09: 1.124
- Product 02: 1.219
- Product 20: 1.310
- Product 05: 1.599
- Product 15: 2.380
- Product 12: 2.400
- Product 21: 3.415
- Product 08: 3.532
- Product 17: 4.533
- Product 03: 8.920
- Product 24: 12.900

yonder

R&D Metrics

24 products measured
2 portfolios
2 had zero code risk
None had zero risk

Risk Type Distribution

Legend: ■ Code Risk ■ Tech Debt Risk ■ Test Risk
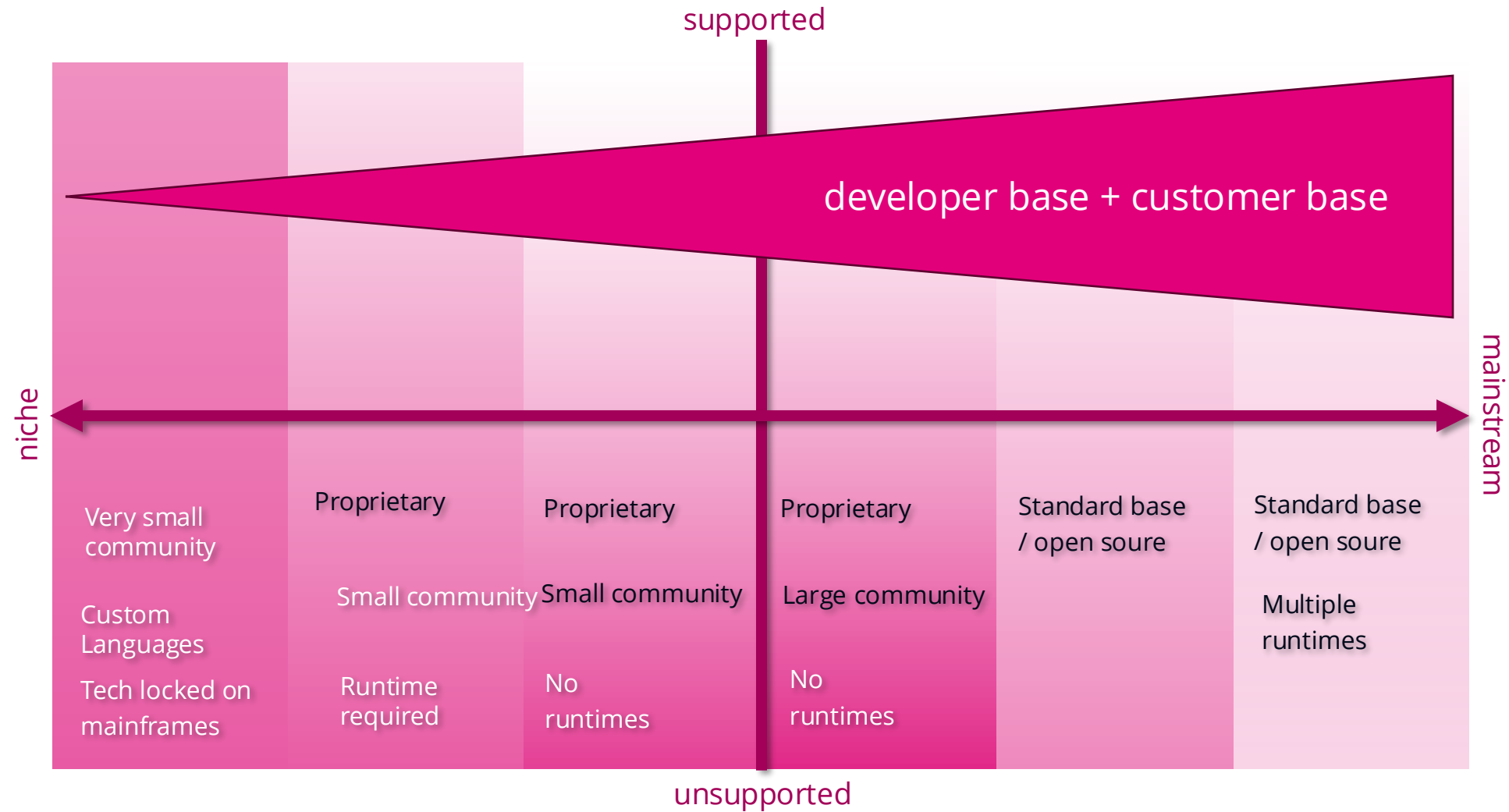
Pie chart values: 1,39% 10% 88,34%

Legend: ■ Code Risk ■ Tech Risk ■ Test Risk

yonder
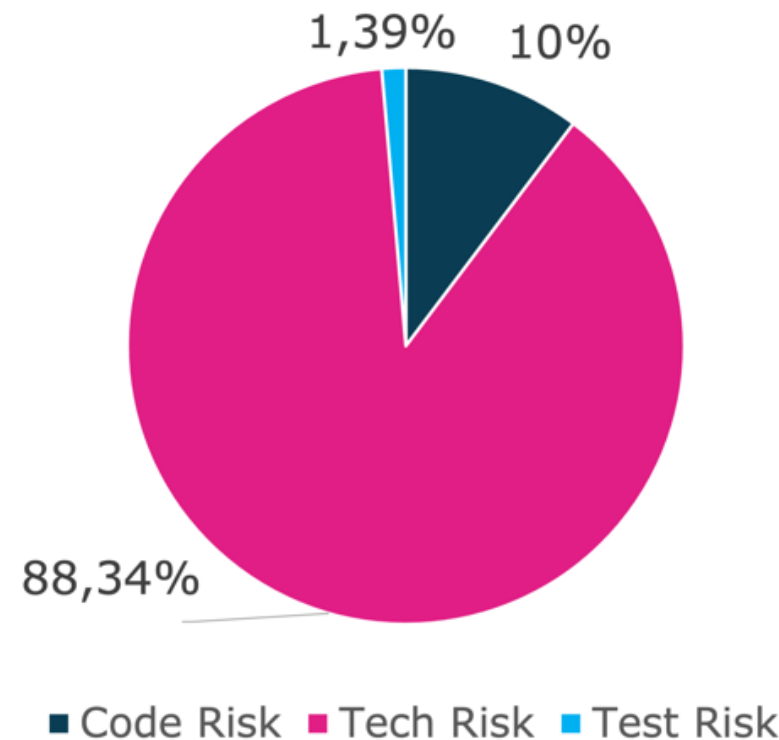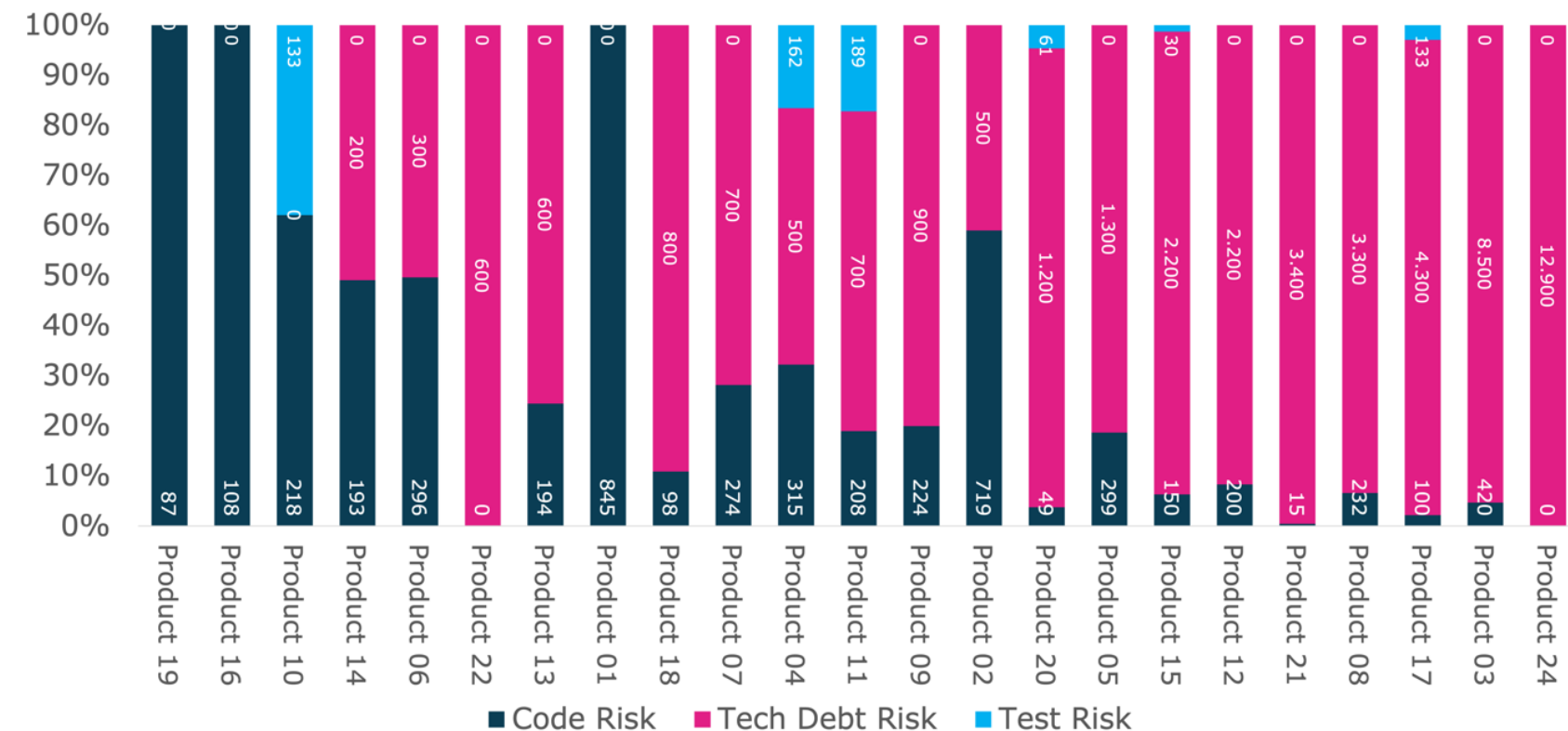
R&D Metrics

Cyclomatic complexity, high number of decisions, the biggest problem.

Code Risk Distribution/Product

Code Risk Composition

yonder

R&D Metrics

supported

developer base + customer base

niche ← → mainstream

Very small community	Proprietary	Proprietary	Proprietary	Standard base / open soure	Standard base / open soure
Custom Languages	Small community	Small community	Large community		Multiple runtimes
Tech locked on mainframes	Runtime required	No runtimes	No runtimes		

unsupported

yonder

Does these results match your expectations?

yonder

2 Impact

What is the impact of code debt is in your portfolio?

yonder

The impact of code debt

Table 2: Average number of Jira defects per file for each Code Health category.

		Healthy	Warning	Alert	All
Jira defects	Avg	0.25	0.94	3.70	0.35
	75%	0.00	1.00	4.00	0.0
	Std	0.90	2.58	6.61	1.43

Relative #Jira defects

15 times more defects compared to Healthy code

Code Health

yonder

The impact of code debt

Relative Time-in-Development

124% longer average compared to Healthy code

Figure 8: Average Time-in-Development (scaled) for resolving a Jira issue per file. The standard errors are depicted as vertical lines.

Average Maximum Time-in-Development

9 times longer average maximum time compared to Healthy code
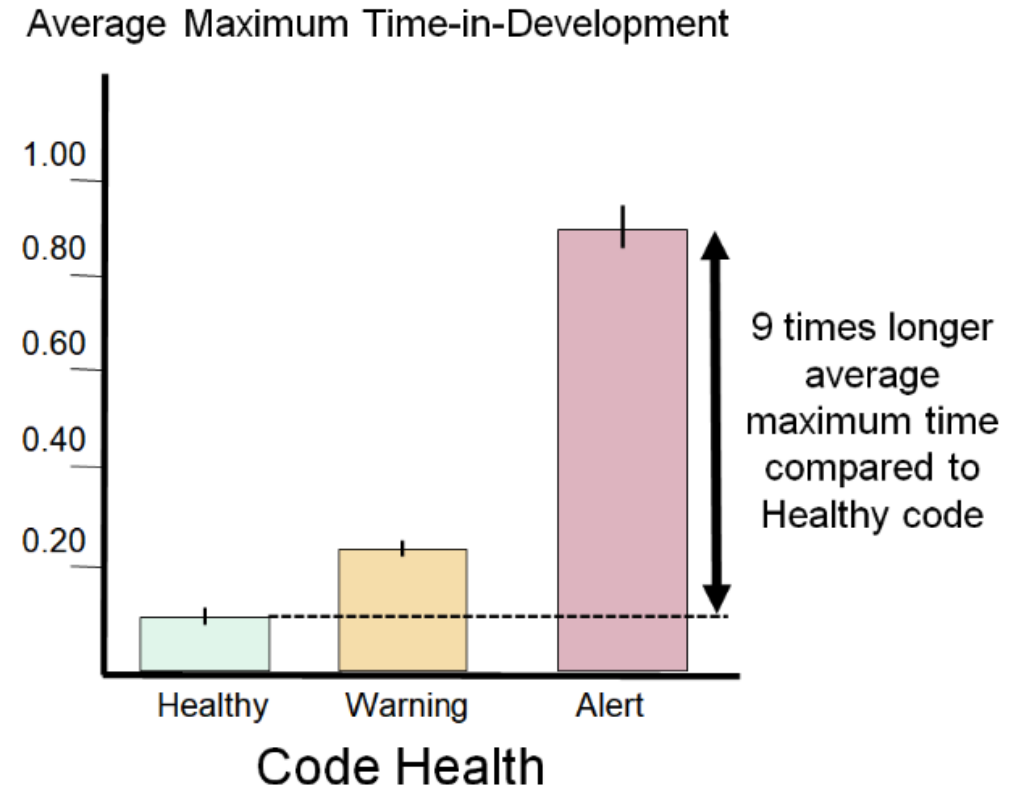
Figure 9: Average maximum Time-in-Development (scaled) for resolving a Jira issue per file. The standard errors are shown as vertical lines.

https://codescene.com/hubfs/web_docs/Business-impact-of-code-quality.pdf
https://arxiv.org/abs/2203.04374

yonder

Tracking planned vs. unplanned work?

yonder

The impact of code debt
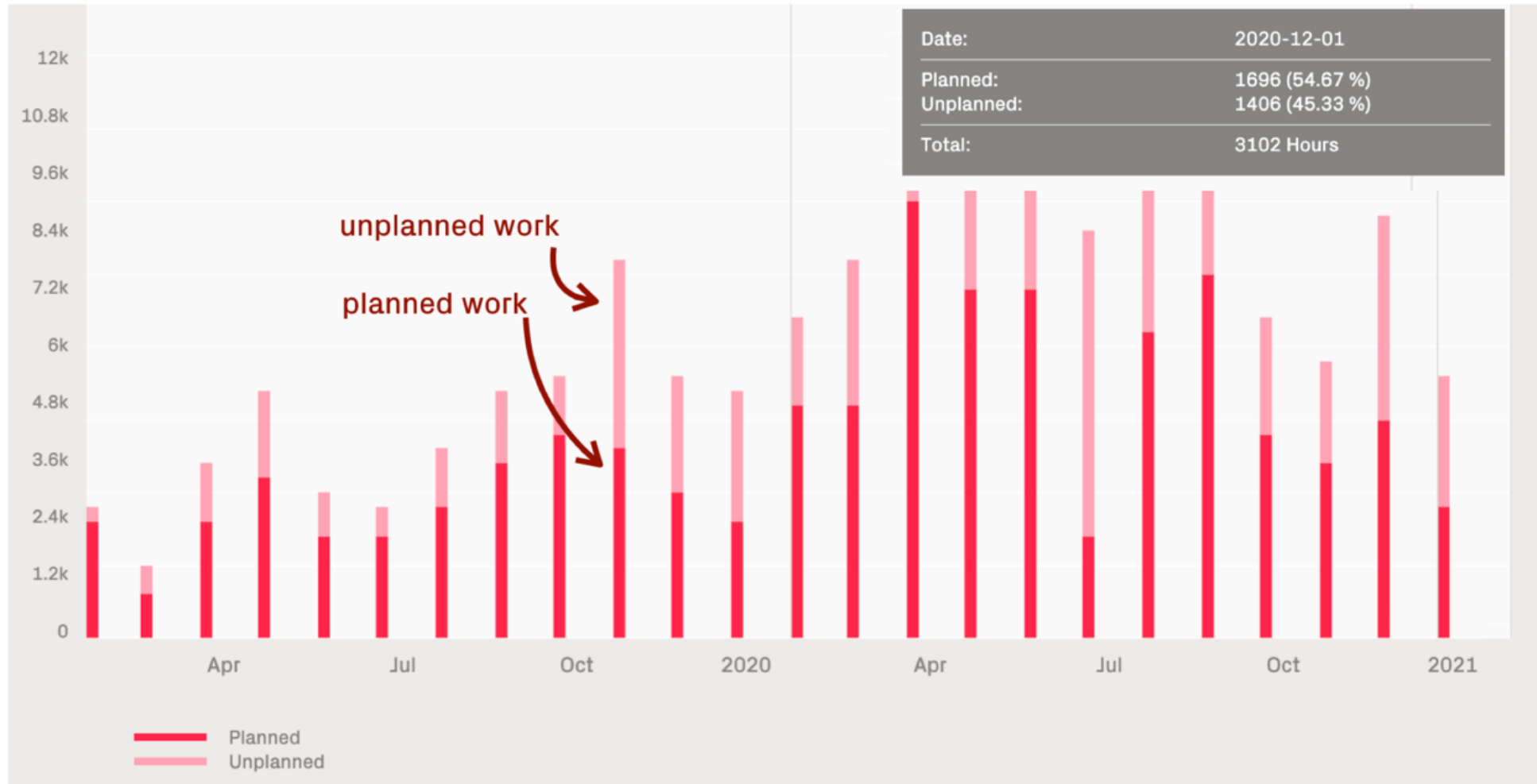
Figure 2. Trend showing the percentage of Unplanned Work over the past year.
On average, 40-50% of the development time is wasted on unplanned work.

yonder

The impact of code debt

Business impact of technical debt

This paper presents an approach to calculating, visualizing, and communicating the costs of technical debt. As shown in this paper, a typical development organization can increase their feature delivery efficiency by at least 25% by managing technical debt.

Software development is rarely sustainable. The average organization wastes 23- 42% of their development time due to technical debt.

Based on data, many organizations pay for 100 developers, but are only getting the output equivalent of 75 developers.

yonder

The impact of code debt
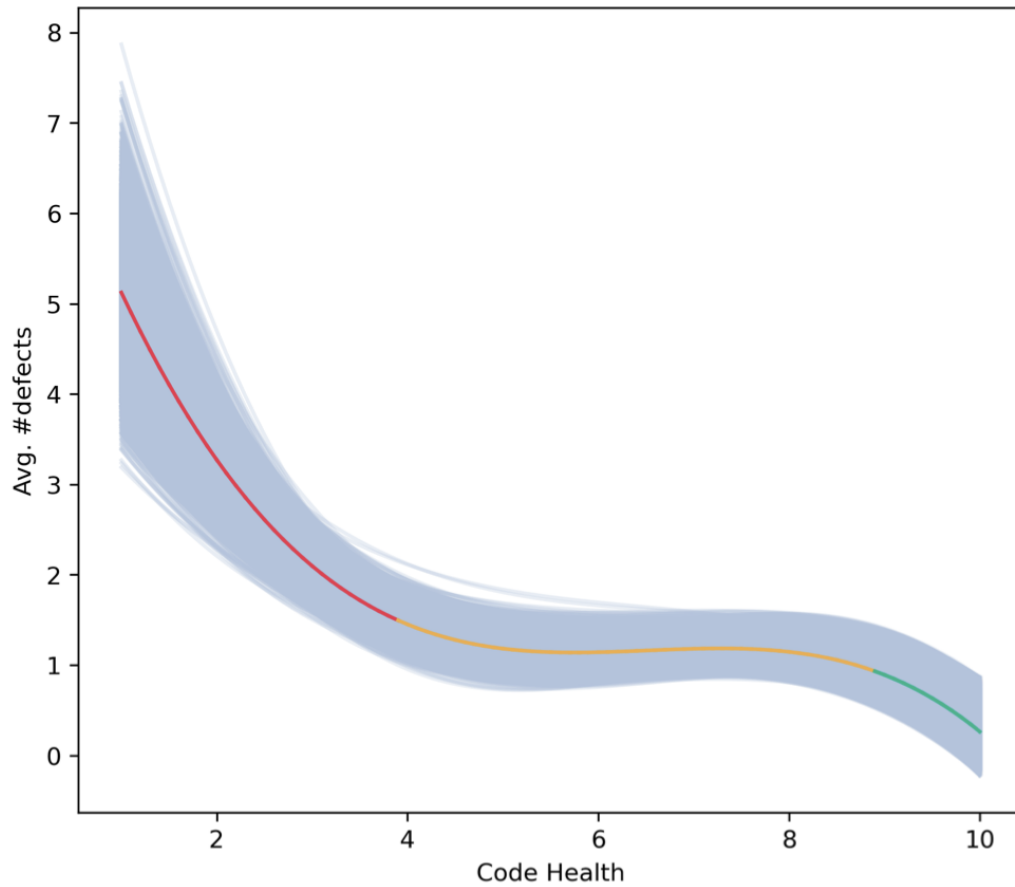
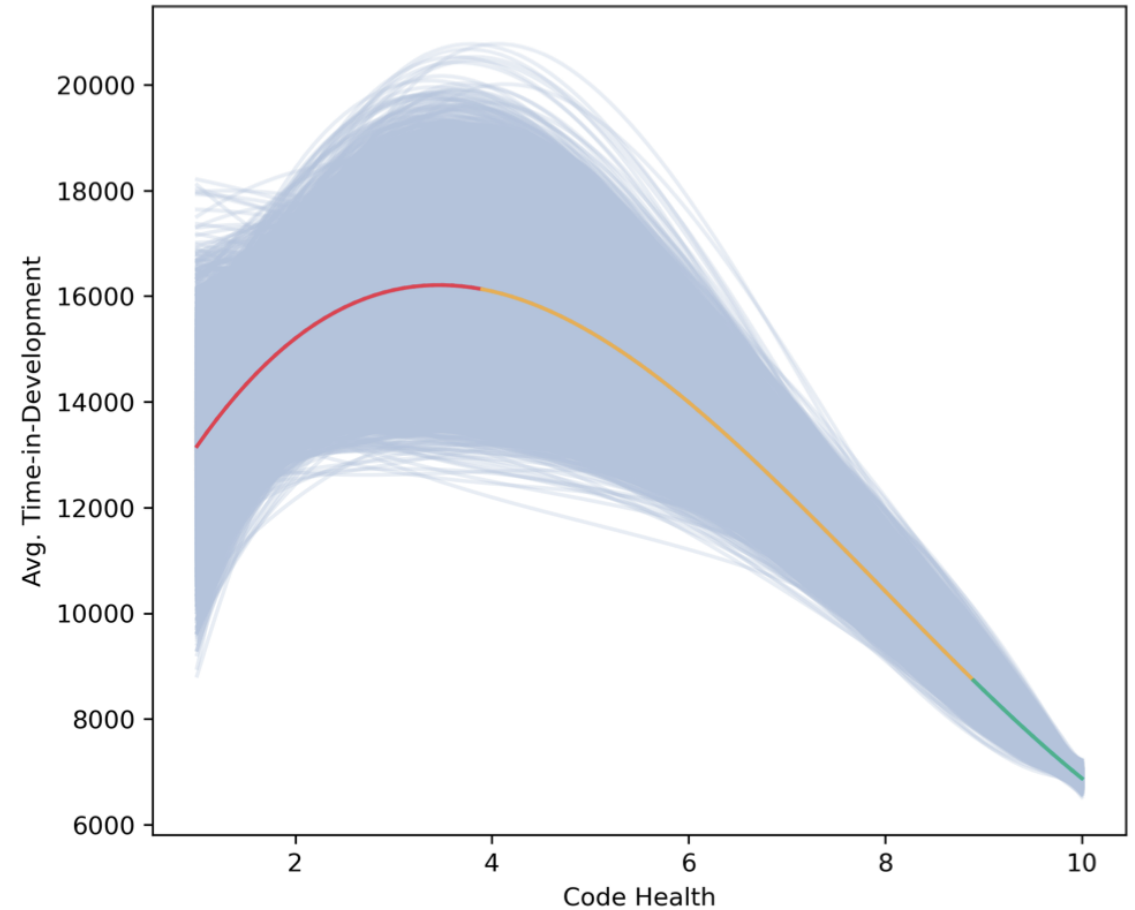**Figure 2: Average defect count per file for different CH.**

https://codescene.com/hubfs/calculate-business-costs-of-technical-debt.pdf
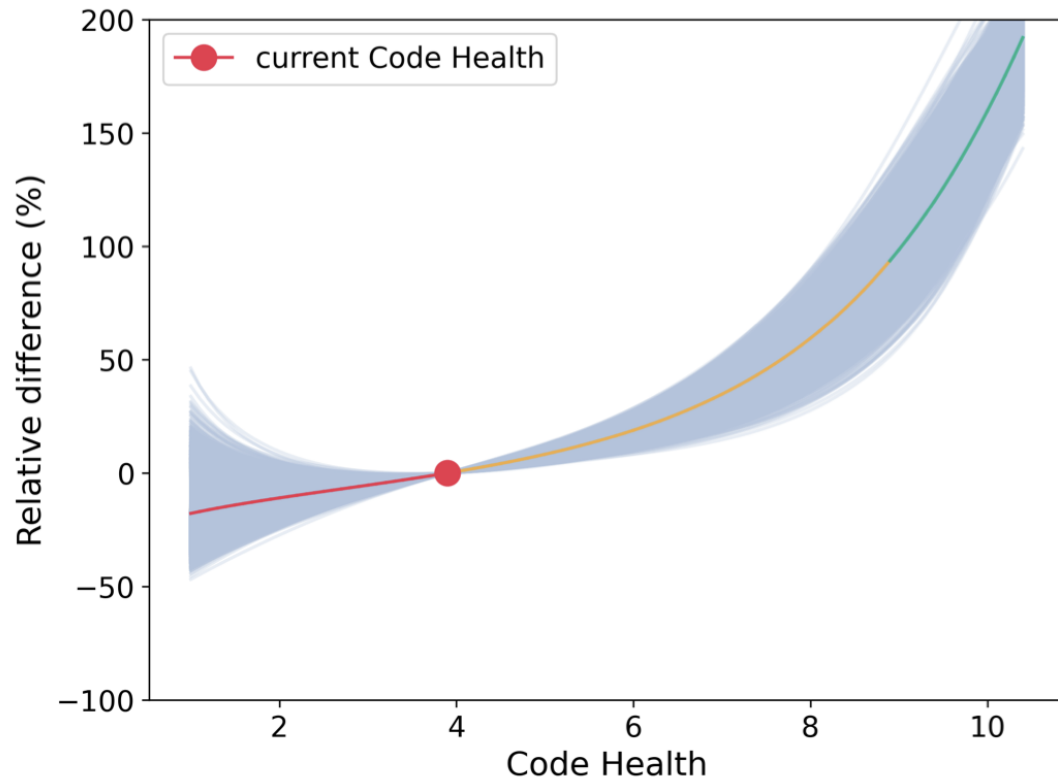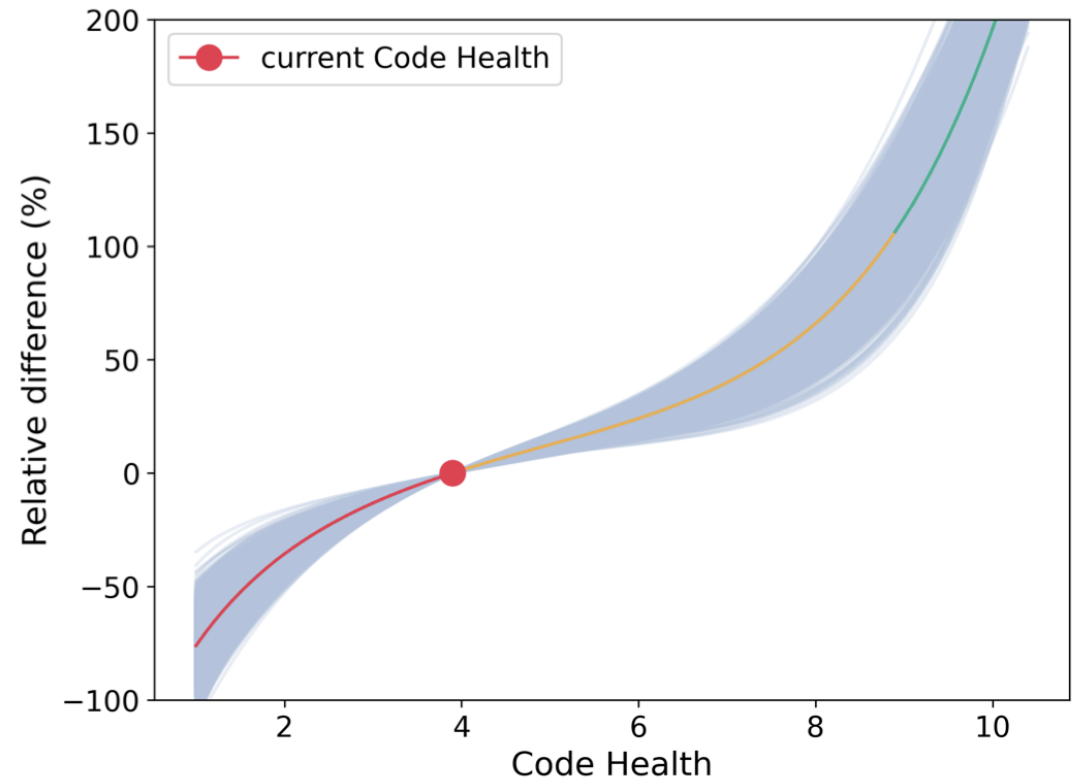https://arxiv.org/pdf/2401.13407v1

Figure 3: Average Time-in-Dev for resolving issues.
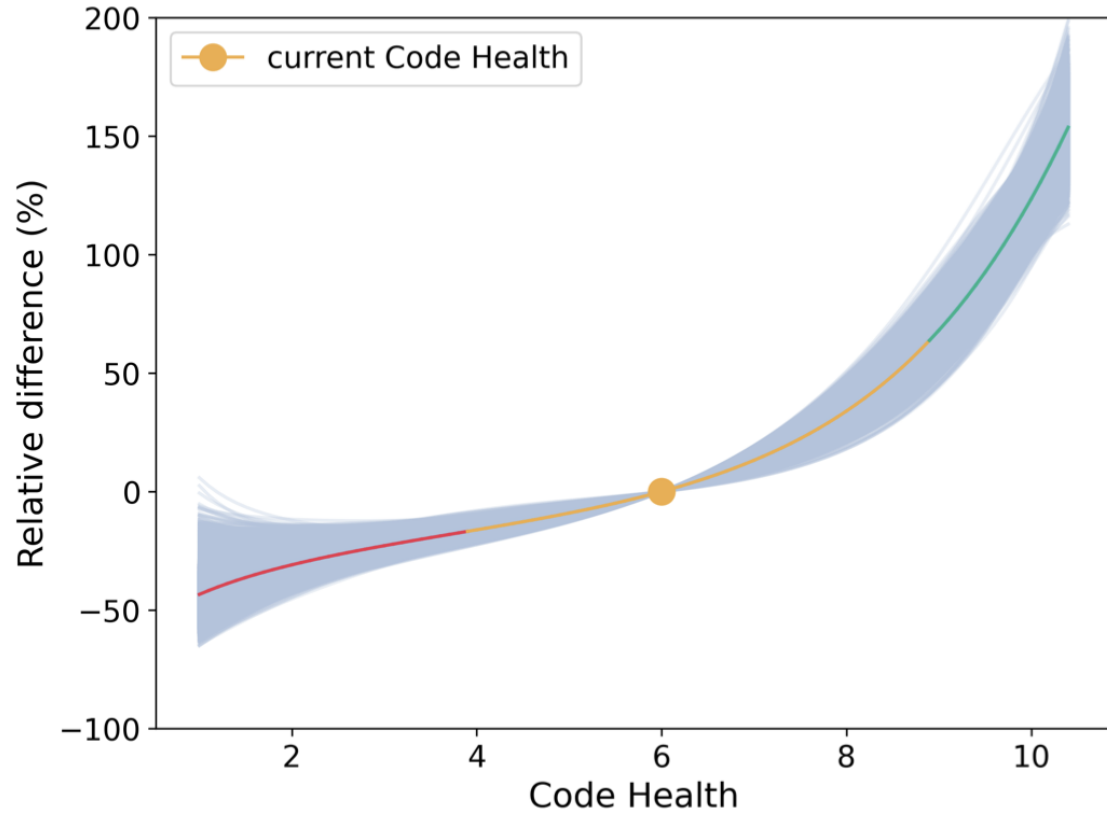
yonder

Increasing, not diminishing returns
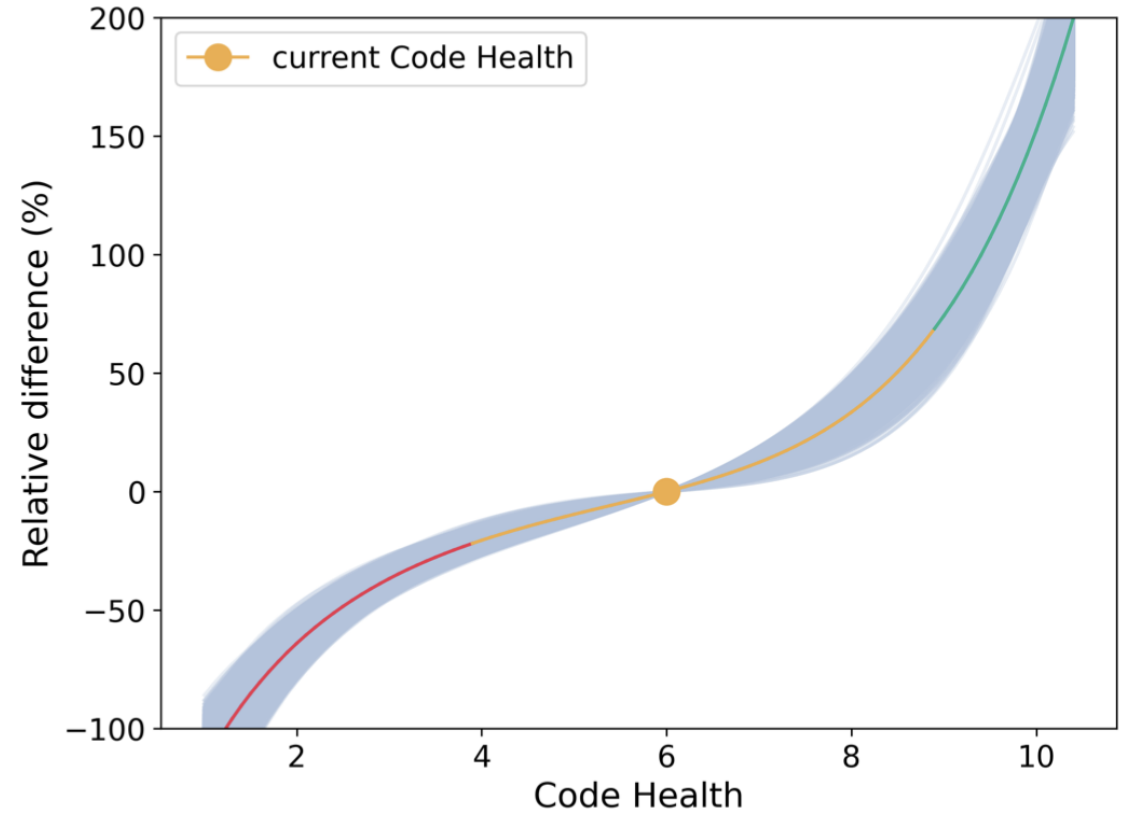

(a) Starting point: $CH_0 = 3.9, u = 0.12$
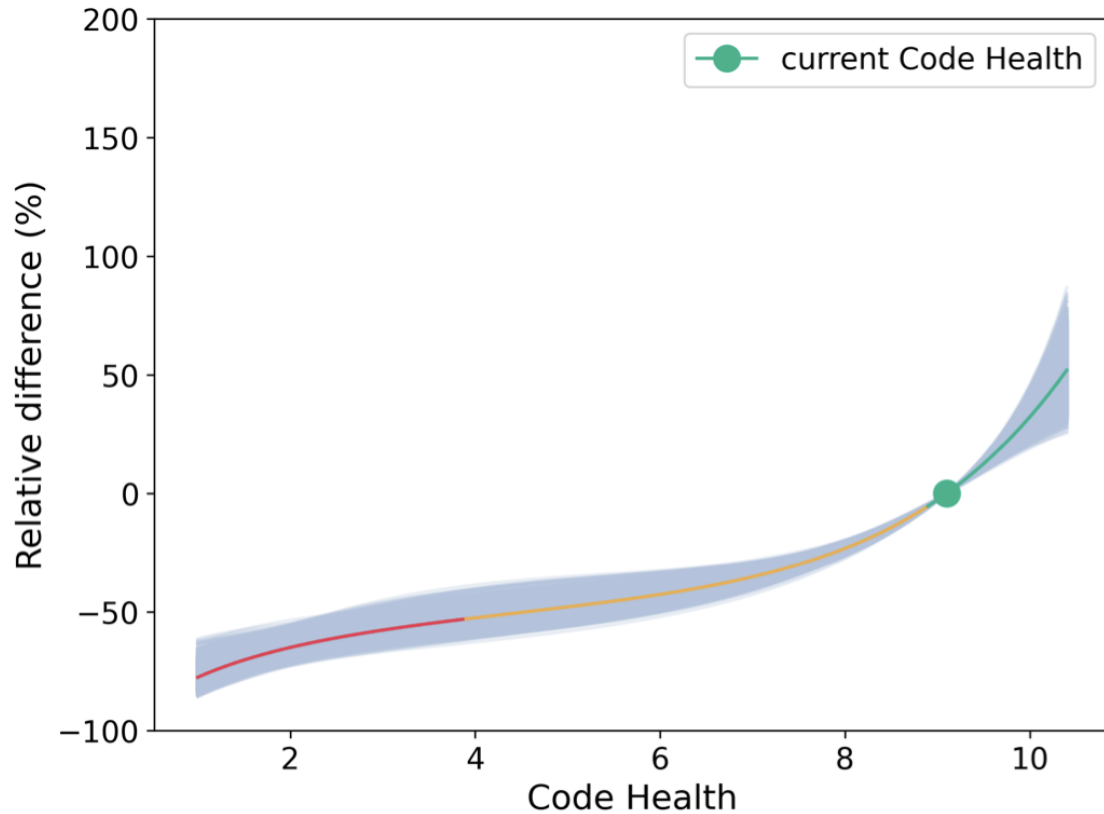
(b) Starting point: $CH_0 = 3.9, u = 0.25$

https://codescene.com/hubfs/calculate-business-costs-of-technical-debt.pdf
https://arxiv.org/pdf/2401.13407v1

yonder

Increasing, not diminishing returns

(c) Starting point: $CH_0 = 6.0, u = 0.12$

(d) Starting point: $CH_0 = 6.0, u = 0.25$

https://codescene.com/hubfs/calculate-business-costs-of-technical-debt.pdf
https://arxiv.org/pdf/2401.13407v1

yonder

Increasing, not diminishing returns

(e) Starting point: $CH_0 = 9.1$, $u = 0.12$
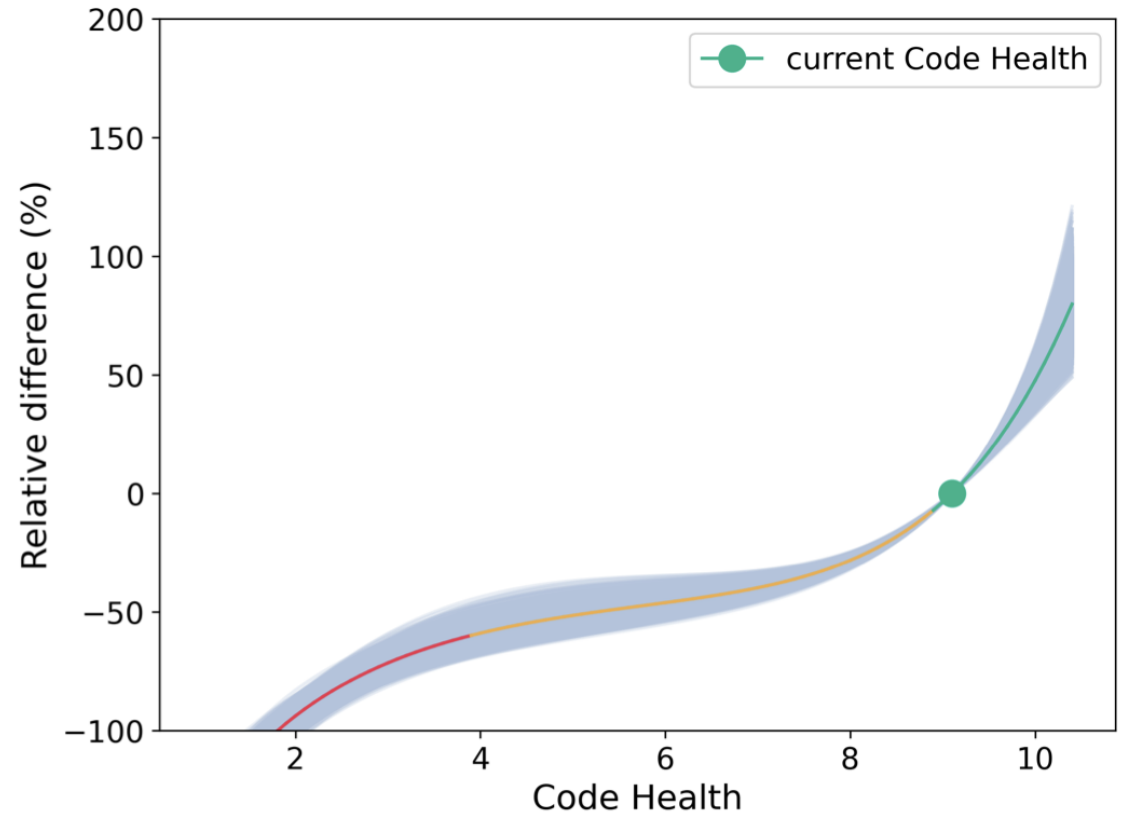
(f) Starting point: $CH_0 = 9.1$, $u = 0.25$

yonder

Should we do, something about Technical Debt?

yonder

thank you

yonder