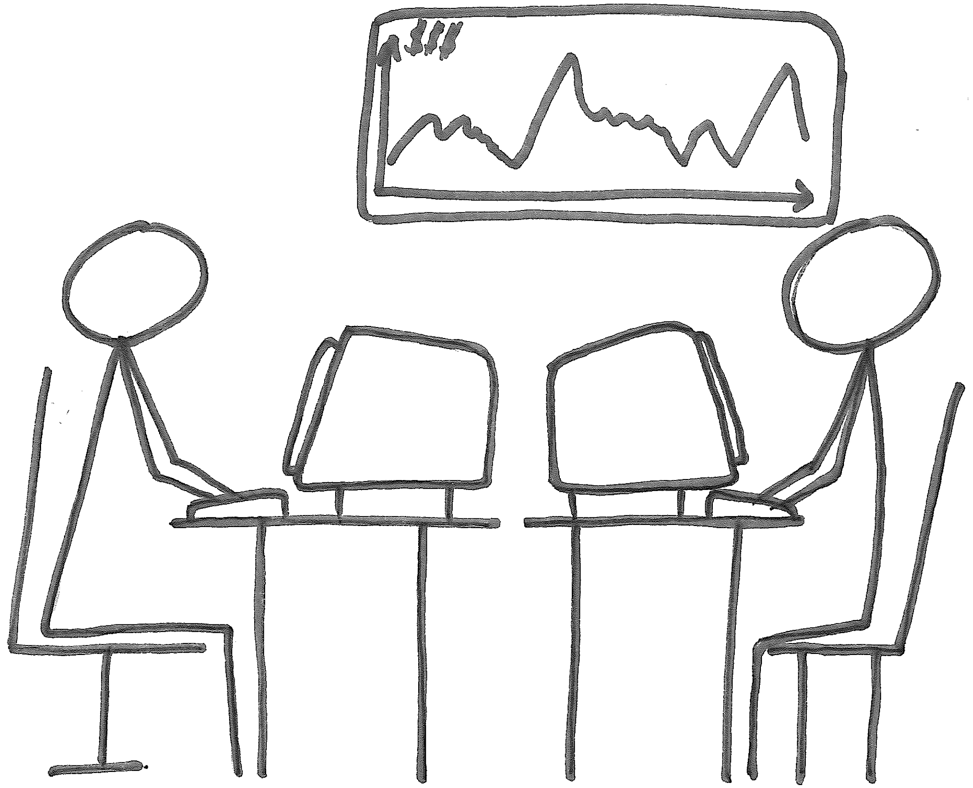
TECHNICAL DEBT

Likely not what you think it is...

TERMINOLOGY

What are we talking about?

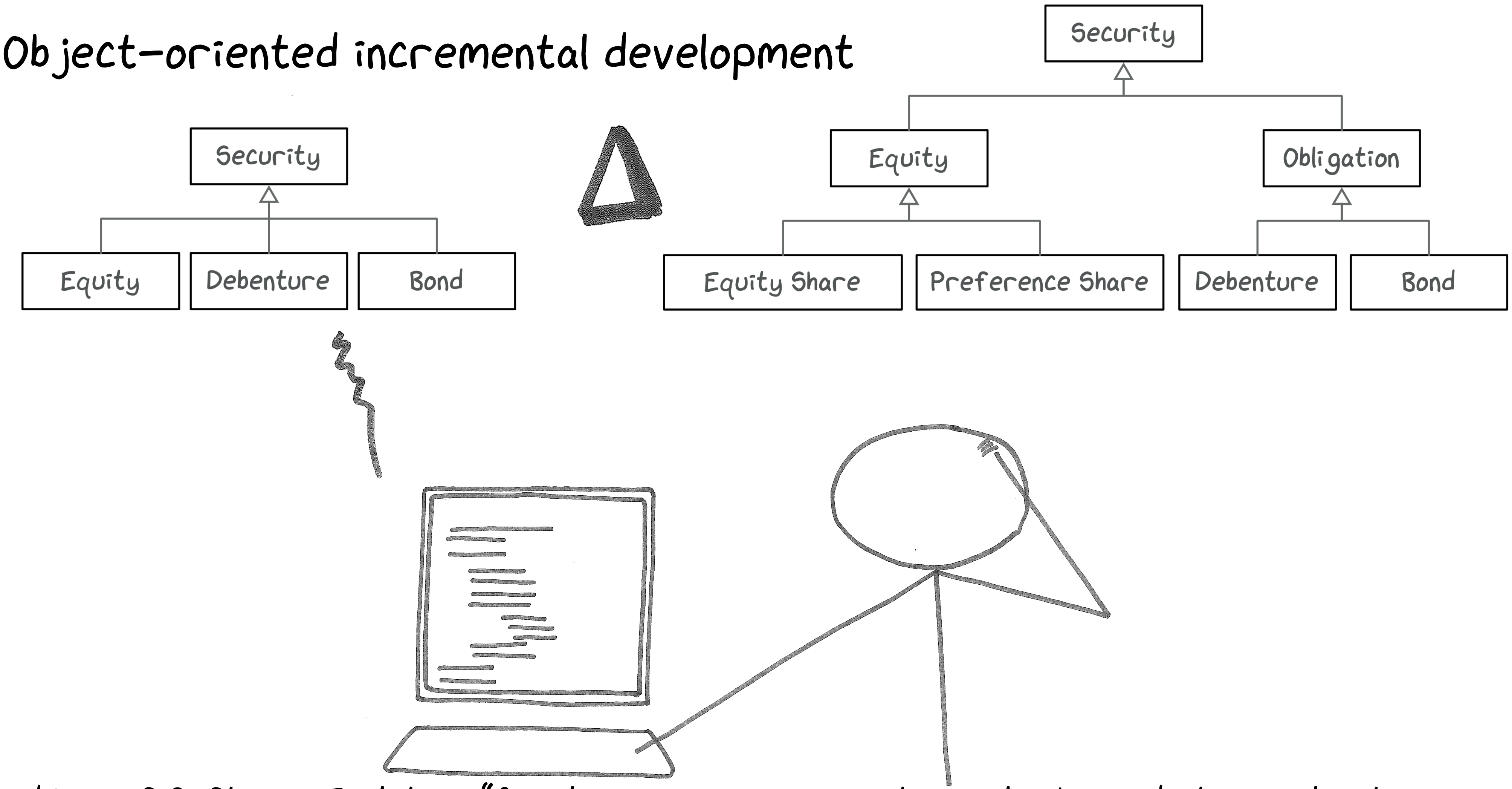
Early 1990s: Development of WyCash



The Metaphor is Born

"Shipping first time code is like going into **debt**. A little debt speeds development so long as it is paid back promptly with a rewrite. [...] The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt."[1]

Object-oriented incremental development



Second Law of Software Evolution: "As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it." [2]

Since then: Babylonian confusion of the metaphor

"The concept of TD contextualizes problems faced during software evolution considering the tasks that are not carried out adequately during software development." [3]

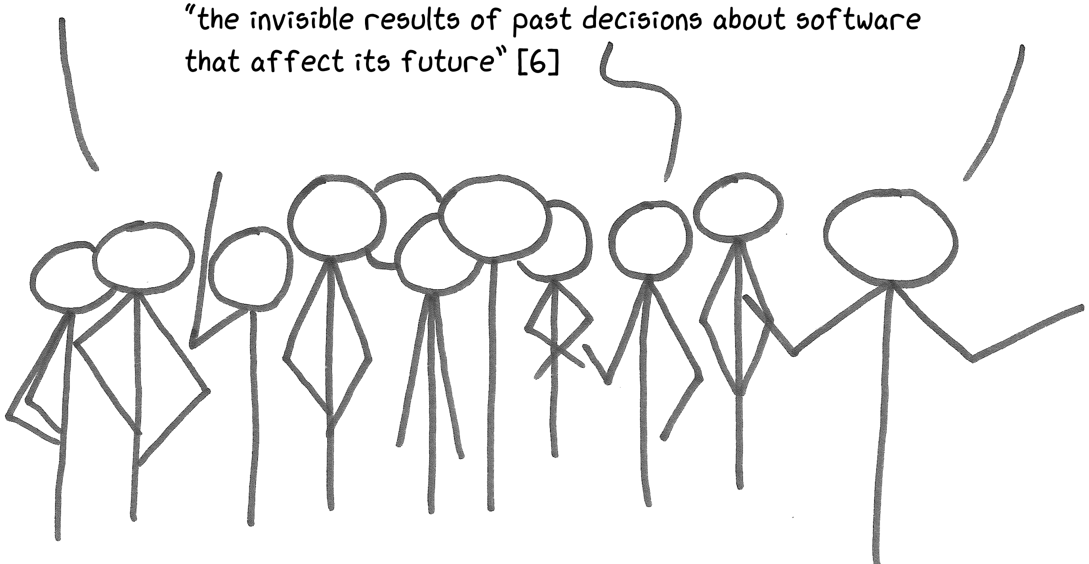
"those internal software development tasks chosen to be delayed, but that run a risk of causing future problems if not done eventually" [4]

"a metaphor for the accumulation of unresolved issues in a software project" [5]

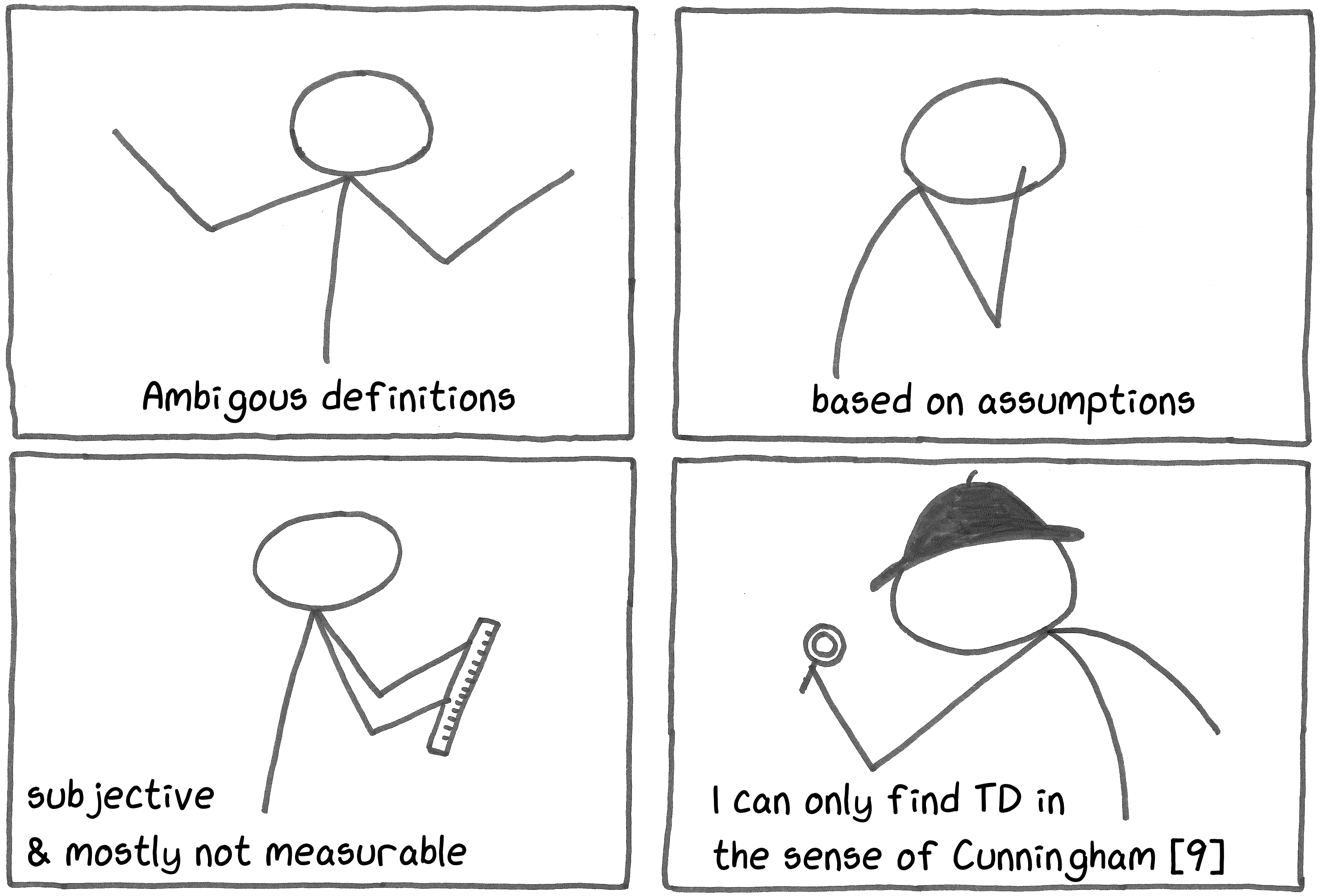
"TD is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. TD presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability." [8]

"the eventual financial consequences of trade-offs between shrinking product time to market and poorly specifying, or implementing a software product, throughout all development phases" [7]

"the invisible results of past decisions about software that affect its future" [6]

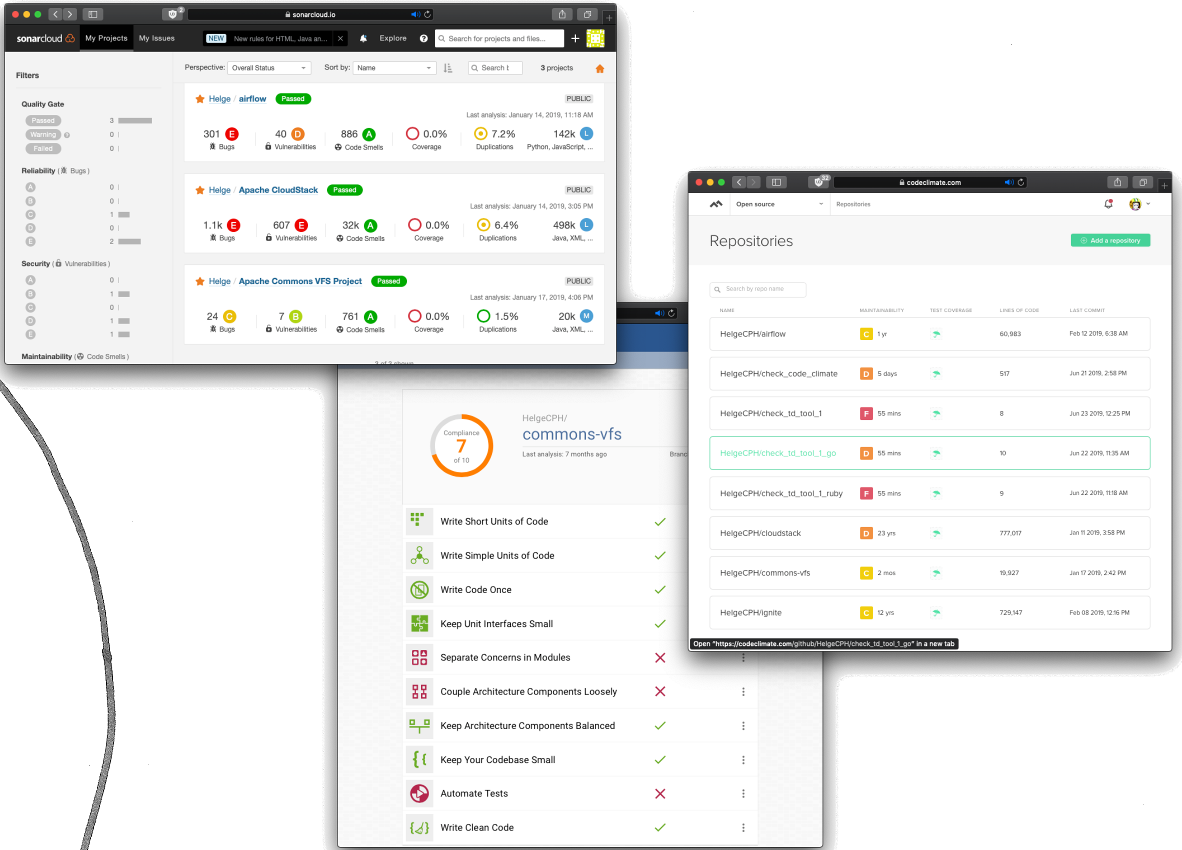


What is the problem?



TOOLS

What do they assess?


sonarcloud.io
sonarcloud
My Projects
My Issues
NEW New rules for HTML, Java an...
Explore
Search for projects and files...
Filters
Perspective: Overall Status
Sort by: Name
Search
3 projects

Quality Gate	
Passed	3
Warning	0
Failed	0

Reliability (Bugs)	
A	0
B	0
C	1
D	0
E	2

Security (Vulnerabilities)	
A	0
B	1
C	0
D	1
E	1

Maintainability (Code Smells)

Helge / airflow Passed PUBLIC
Last analysis: January 14, 2019, 11:18 AM

Bugs	Vulnerabilities	Code Smells	Coverage	Duplications	
301 E	40 D	886 A	0.0%	7.2%	142k L Python, JavaScript, ...

Helge / Apache CloudStack Passed PUBLIC
Last analysis: January 14, 2019, 3:05 PM

Bugs	Vulnerabilities	Code Smells	Coverage	Duplications	
1.1k E	607 E	32k A	0.0%	6.4%	498k L Java, XML, ...

Helge / Apache Commons VFS Project Passed PUBLIC
Last analysis: January 17, 2019, 4:06 PM

Bugs	Vulnerabilities	Code Smells	Coverage	Duplications	
24 C	7 B	761 A	0.0%	1.5%	20k M Java, XML, ...

Compliance 7 of 10

HelgeCPH/
commons-vfs
Last analysis: 7 months ago

Write Short Units of Code	✓
Write Simple Units of Code	✓
Write Code Once	✓
Keep Unit Interfaces Small	✓
Separate Concerns in Modules	✗
Couple Architecture Components Loosely	✗
Keep Architecture Components Balanced	✓
Keep Your Codebase Small	✓
Automate Tests	✗
Write Clean Code	✓

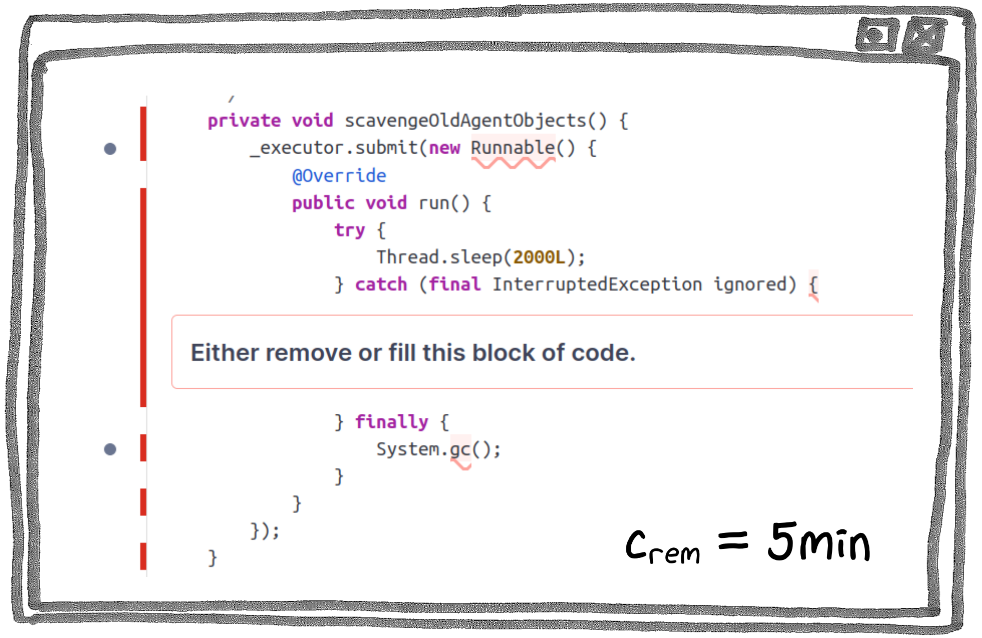
Repositories

NAME	MAINTAINABILITY	TEST COVERAGE	LINES OF CODE	LAST COMMIT
HelgeCPH/airflow	C 1 yr		60,983	Feb 12 2019, 6:38 AM
HelgeCPH/check_code_climate	D 5 days		517	Jun 21 2019, 2:58 PM
HelgeCPH/check_td_tool_1	F 55 mins		8	Jun 23 2019, 12:25 PM
HelgeCPH/check_td_tool_1_go	D 55 mins		10	Jun 22 2019, 11:35 AM
HelgeCPH/check_td_tool_1_ruby	F 55 mins		9	Jun 22 2019, 11:18 AM
HelgeCPH/cloudstack	D 23 yrs		777,017	Jan 11 2019, 3:58 PM
HelgeCPH/commons-vfs	C 2 mos		19,927	Jan 17 2019, 2:42 PM
HelgeCPH/ignite	C 12 yrs		729,147	Feb 08 2019, 12:16 PM

Open "https://codeclimate.com/github/HelgeCPH/check_td_tool_1_go" in a new tab

A Tool Example: SonarQube [10,11,12]

$$\text{Technical Debt Ratio} = \frac{C_{rem}}{C_{line} * LOC}$$



```
private void scavengeOldAgentObjects() {
    _executor.submit(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(2000L);
            } catch (final InterruptedException ignored) {
```

Either remove or fill this block of code.

```
            } finally {
                System.gc();
            }
        }
    });
}
```

$C_{rem} = 5min$

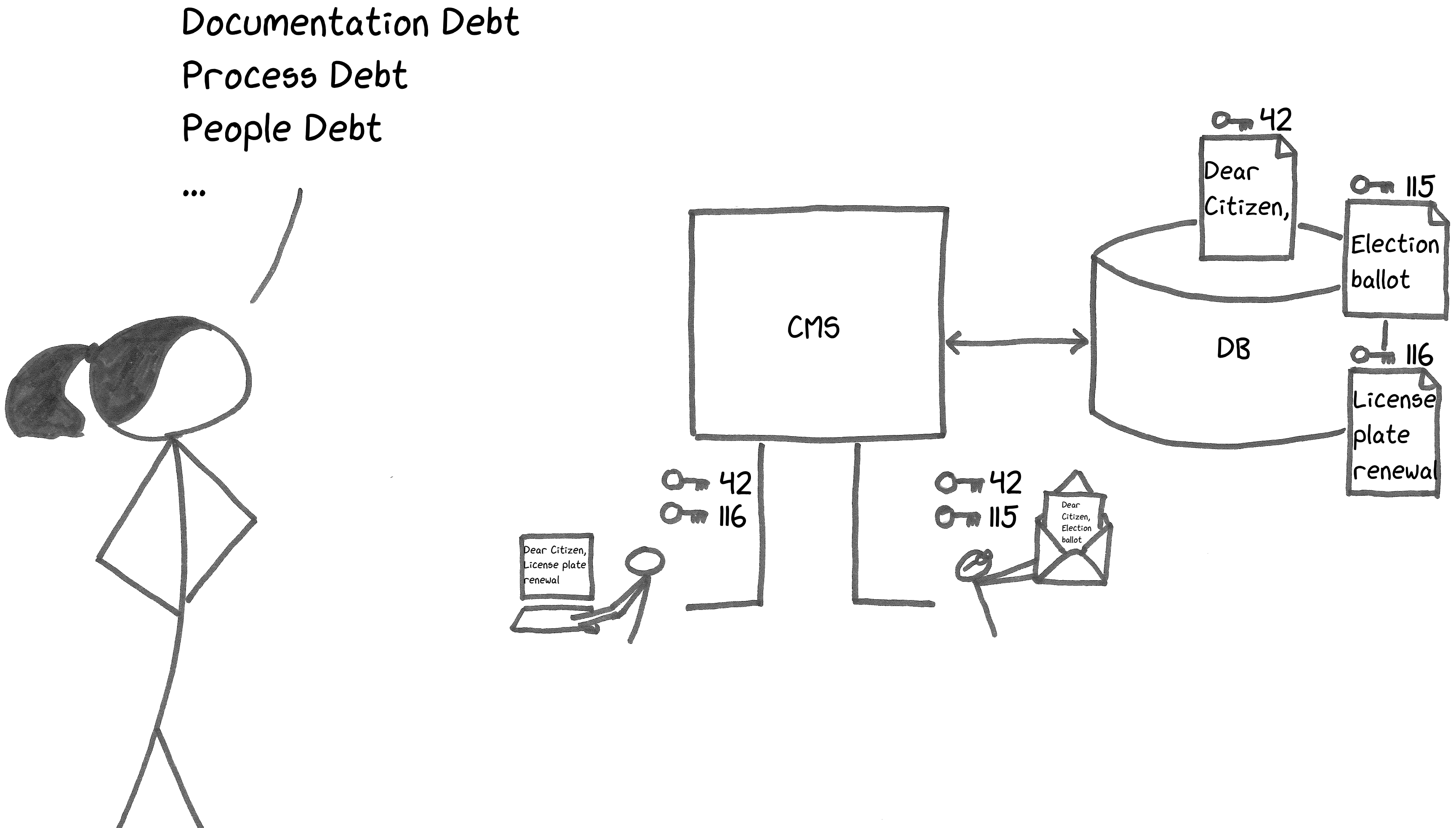
Maintainability Rating =

- A ... $0 < TD <= 0.05$
- B ... $0.06 < TD <= 0.1$
- C ... $0.1 < TD <= 0.2$
- D ... $0.21 < TD <= 0.5$
- E ... $0.51 < TD <= 1$

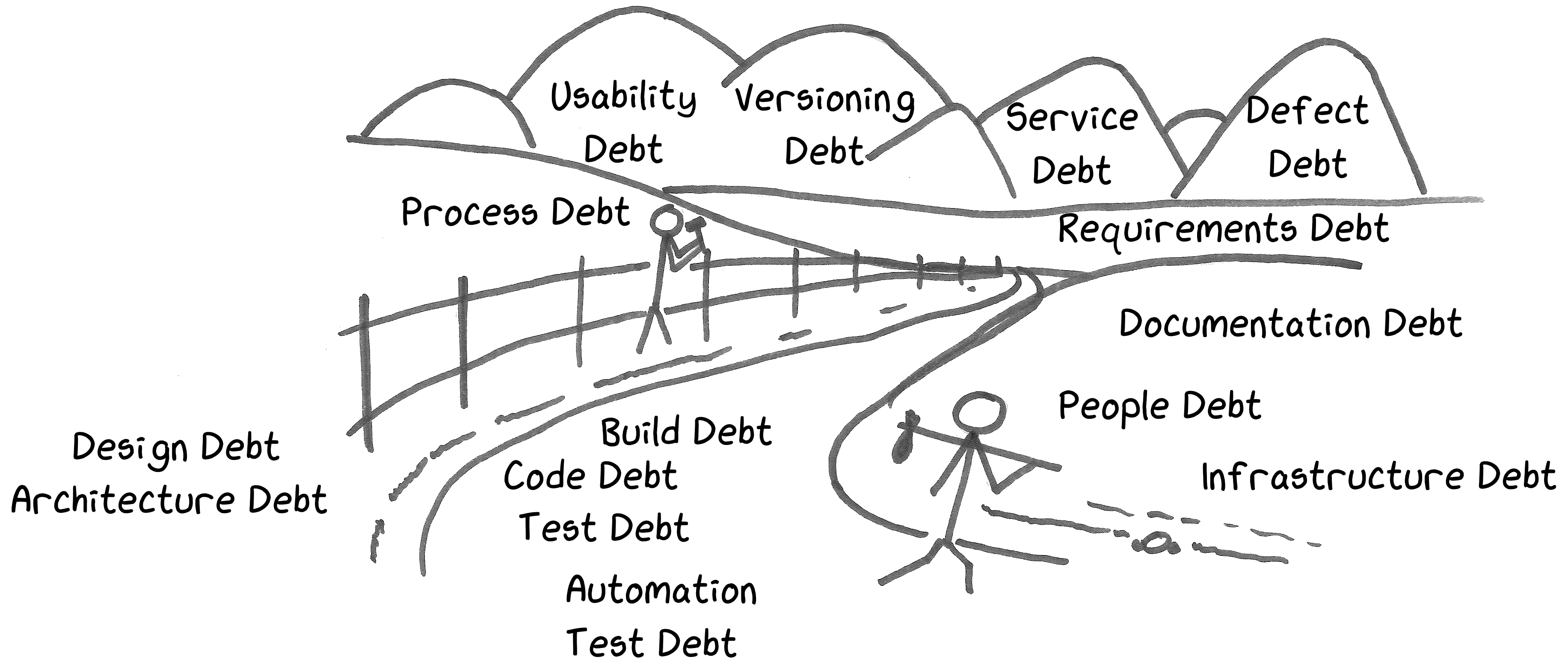
TECHNICAL DEBT

Something more than "technical"...

"Technical debt" in case of a content management system [13]



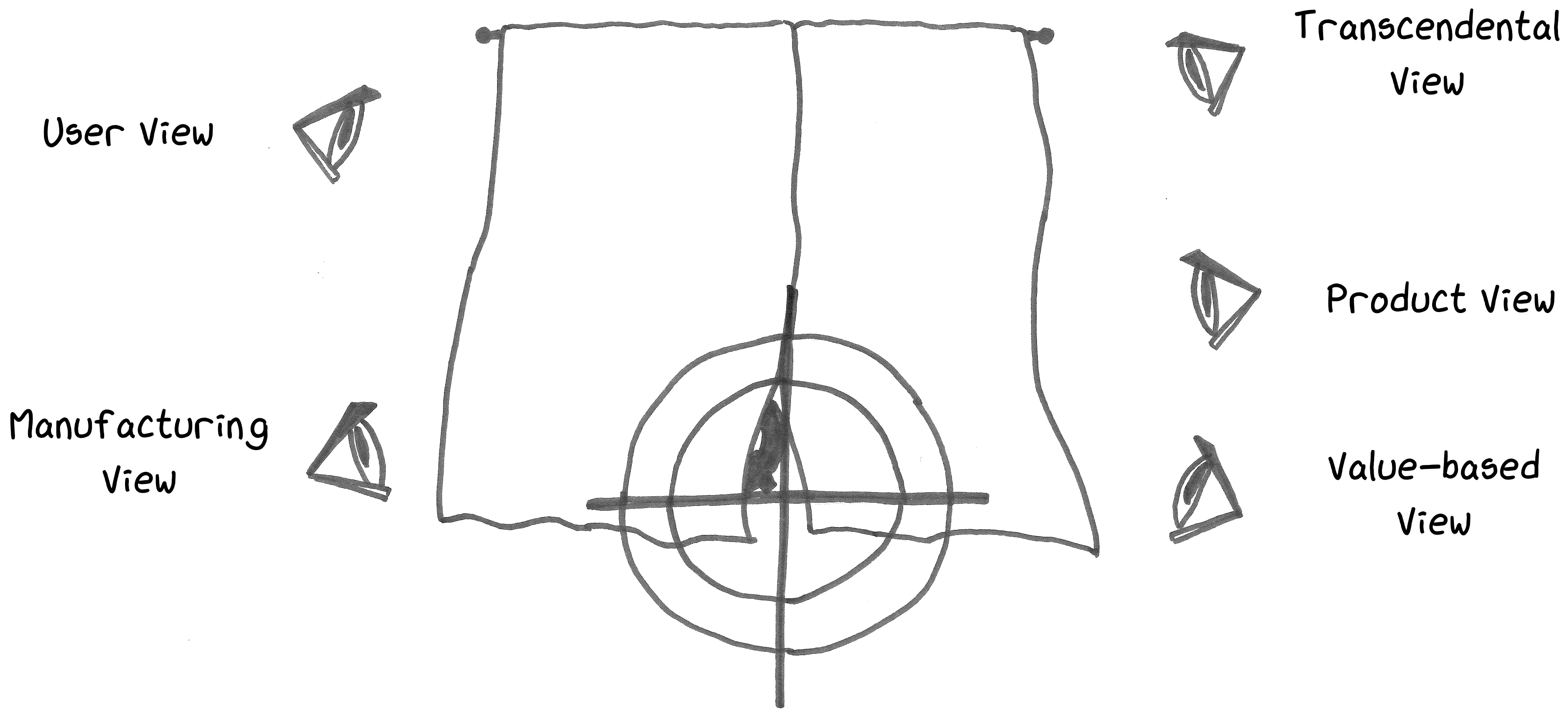
The "technical debt landscape" [14]



WHAT NOW???

Can't I use the concept for anything?

Software Quality: The Elusive Target [15]



CONCLUSION & RECOMMENDATIONS

A metaphor should not be basis for IT project management.

- Technical Debt is neither a well-defined term nor a well-understood concept. Therefore, use precise software qualities for assessment and communication.
- Be aware: incremental software development creates always technical debt.
- Plan, budget, and schedule refactoring work.

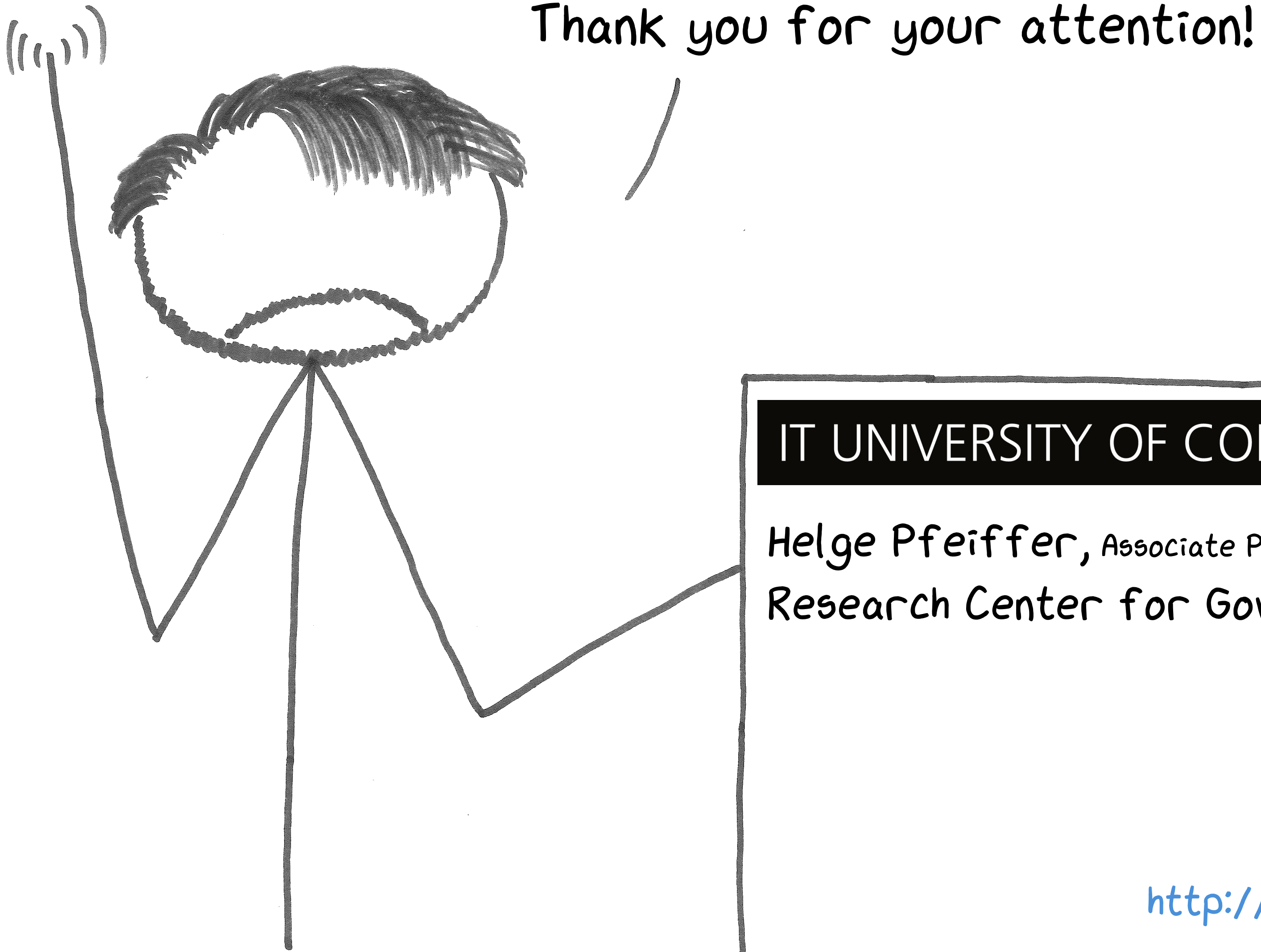
If interested in assessment of software quality:

- Precisely define *software quality* and its assessment so that all involved stakeholders share a common understanding of quality.
- This is inline with what for example the ISO/IEC 25000 series of standards state about comprehensive specification and evaluation of software quality:

"... can be achieved by defining the necessary and desired quality characteristics associated with the stakeholders' goals and objectives for the system. This includes quality characteristics related to the software system and data as well as the impact the system has on its stakeholders. It is important that the quality characteristics are specified, measured, and evaluated whenever possible using validated or widely accepted measures and measurement methods." [16]

Tool-based automatic assessment of software quality and technical debt

- Technical debt *"is generally not detectable by static analysis [since] thoughts are stubbornly hidden from static analysis tools"* [17]
- Be aware of what tools measure and if that aligns with your conception of *software quality*.
- Do not rely on automatic assessment results at face value.
- Be aware of Goodhart's Law: *"When a measure becomes a target, it ceases to be a good measure."* [18]



IT UNIVERSITY OF COPENHAGEN

Helge Pfeiffer, Associate Professor

Research Center for Government IT

ropf@itu.dk

+45-7218 5102

http://itu.dk/~ropf/

REFERENCES

[1] Cunningham, Ward. "The WyCash portfolio management system." ACM Sigplan Oops Messenger 4.2 (1992): 29-30. (link)
[2] Lehman, Meir M. "Laws of program evolution-rules and tools for programming management." Proc. of the Infotech State of the Art Conf., Why Software Projects Fail? Program Press, 1978. Vol. 11. 1978.
[3] Rios, Nicolli, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners." Information and Software Technology 102 (2018): 117-145. (link)
[4] Alves, Nicolli SR, et al. "Identification and management of technical debt: A systematic mapping study." Information and Software Technology 70 (2016): 100-121. (link)
[5] Birchall, Chris. Re-engineering legacy software. Simon and Schuster, 2016.
[6] Kruchten, Philippe, et al. "Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt." ACM SIGSOFT Software Engineering Notes 38.5 (2013): 51-54. (link)
[7] Ampatzoglou, Areti, et al. "The financial aspect of managing technical debt: A systematic literature review." Information and Software Technology 64 (2015): 52-73. (link)
[8] Avgeriou, Paris, et al. "Managing technical debt in software engineering (dagstuhl seminar 16162)." (2016). (link)
[9] Pfeiffer, Rolf-Helge. "Searching for Technical Debt-An Empirical, Exploratory, and Descriptive Case Study." 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2022. (link)
[10] Pfeiffer, Rolf-Helge, and Mircea Lungu. "Technical Debt and Maintainability: How do tools measure it?." arXiv preprint arXiv:2202.13464 (2022). (link)
[11] Pfeiffer, Rolf-Helge, and Jon Aaen. "Tools for monitoring software quality in information systems development and maintenance: five key challenges and a design proposal." International Journal of Information Systems and Project Management 12.1 (2024): 19-40. (link)
[12] Letouzey, Jean-Louis, and Michel Ilkiewicz. "Managing technical debt with the sqale method." IEEE software 29.6 (2012): 44-51. (link)
[13] Nielsen, Mille Edith, and Christian Østergaard Madsen. "Stakeholder influence on technical debt management in the public sector: An embedded case study." Government Information Quarterly 39.3 (2022): 101706. (link)
[14] Rios, Nicolli, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners." Information and Software Technology 102 (2018): 117-145. (link)
[15] Kitchenham, Barbara, and Shari Lawrence Pfleeger. "Software quality: the elusive target [special issues section]." IEEE software 13.1 (1996): 12-21. (link)
[16] “Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SquaRE,” International Organization for Standardization, Geneva, CH, Standard, Mar. 2014.
[17] Fairbanks, George. "Ur-technical debt." IEEE Software 37.4 (2020): 95-98. (link)
[18] Strathern, Marilyn. "‘Improving ratings’: audit in the British University system." European review 5.3 (1997): 305-321.