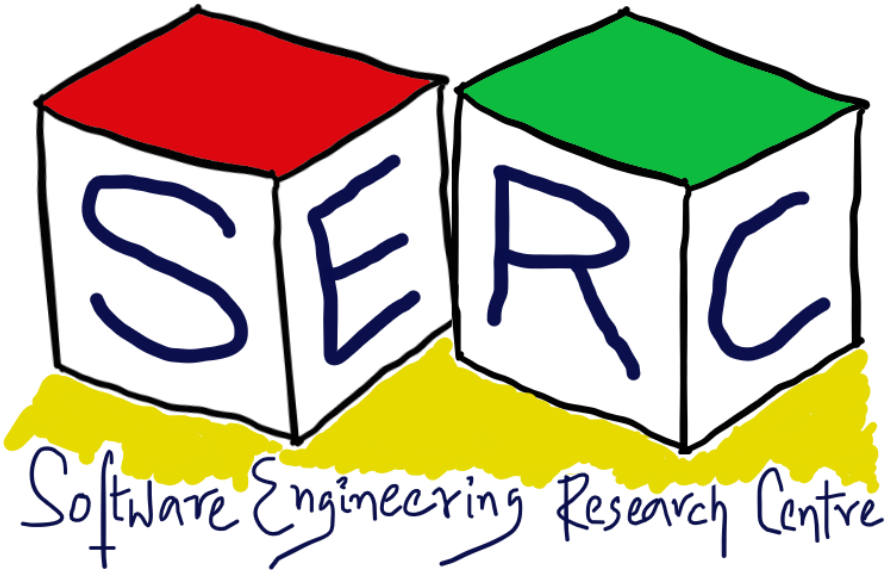
Refactoring: An Introduction

CS6.401 Software Engineering



Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Acknowledgements

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge

-- Karthik Vaidhyanathan

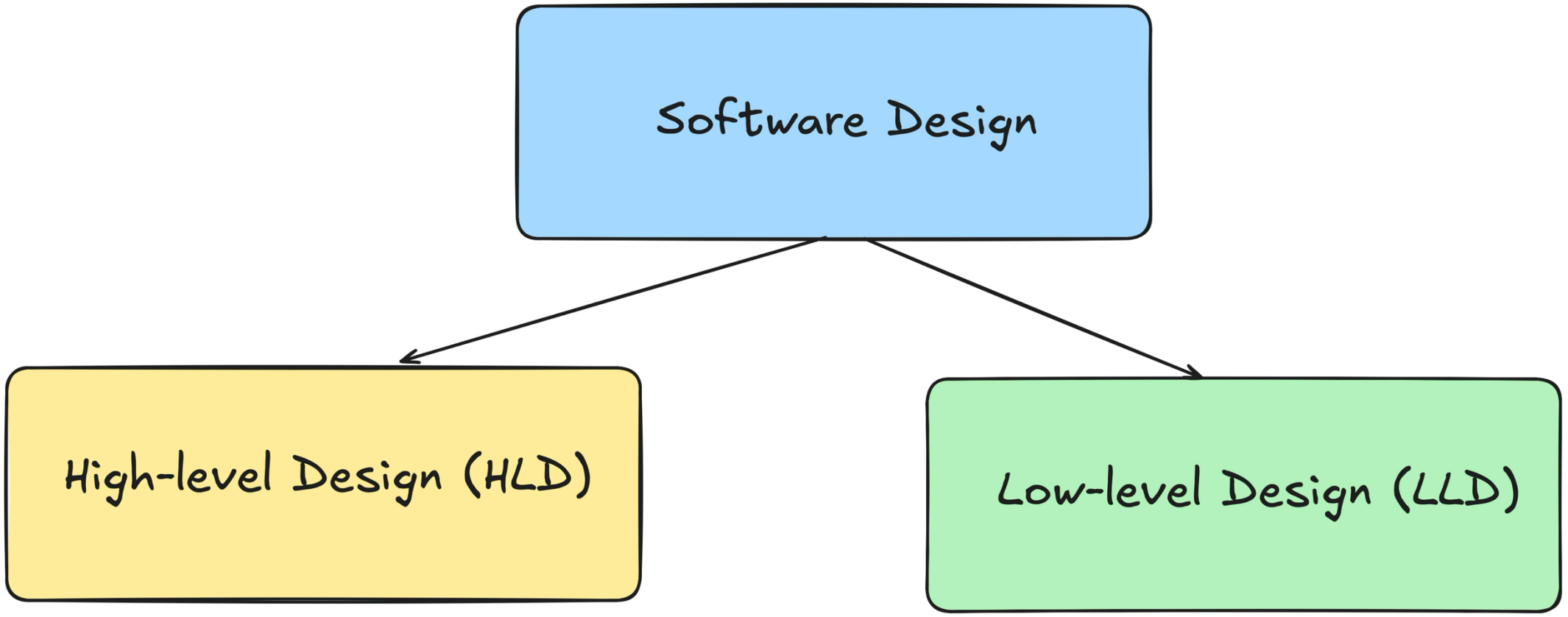
Sources:

1. Refactoring, Improving the design of existing code, Martin Fowler et al., 2000
2. Refactoring for Software design Smells, Girish Suryanarayana et al.
3. martinfowler.com
4. Few articles by Ipek Ozkaya and Robert Nord, SEI, CMU

Software Design

The function of good software is to make the complex appear to be simple.

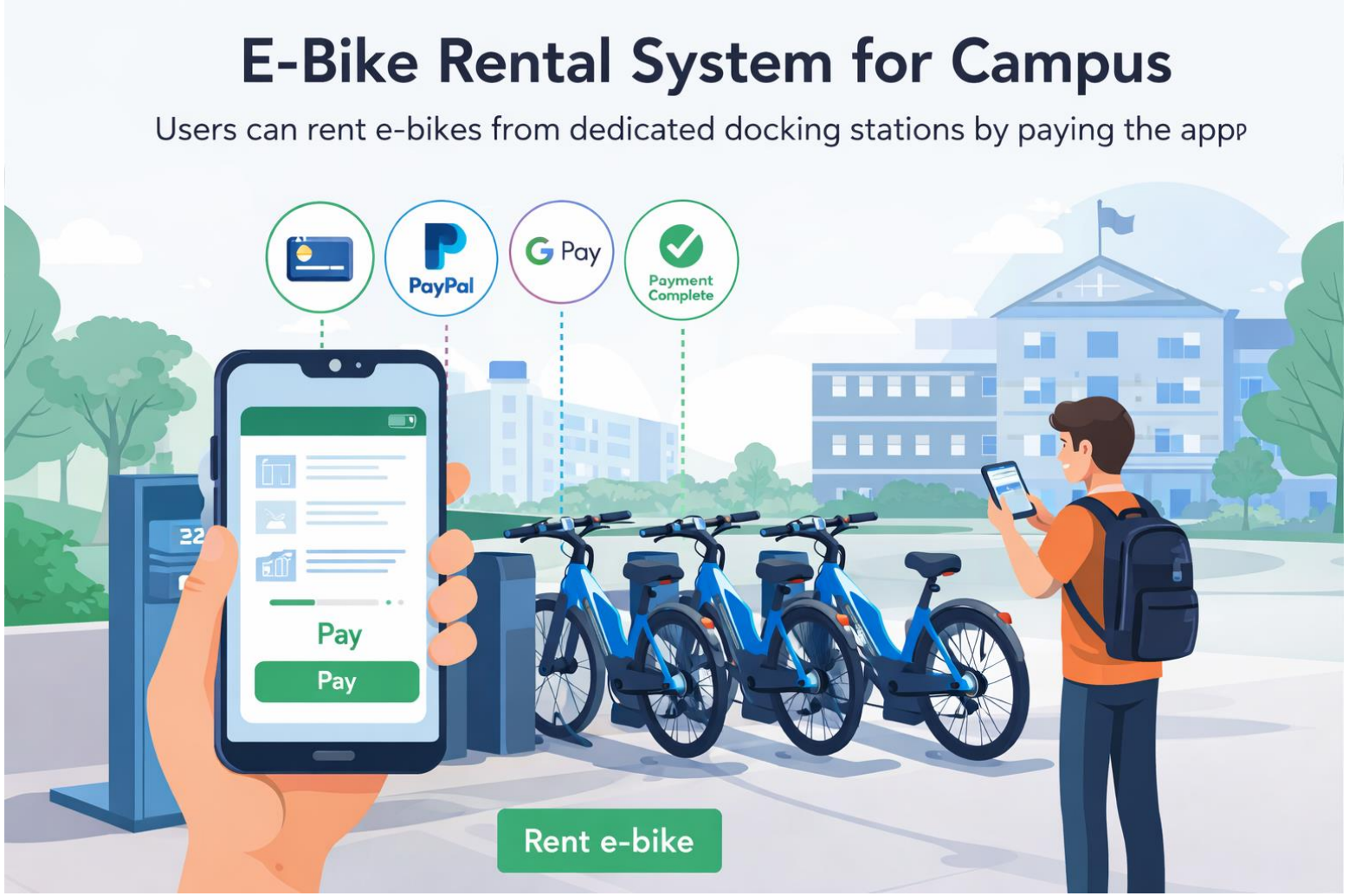
- Grady Booch



As an E-type system evolves, its complexity increases unless work is done to maintain or reduce it

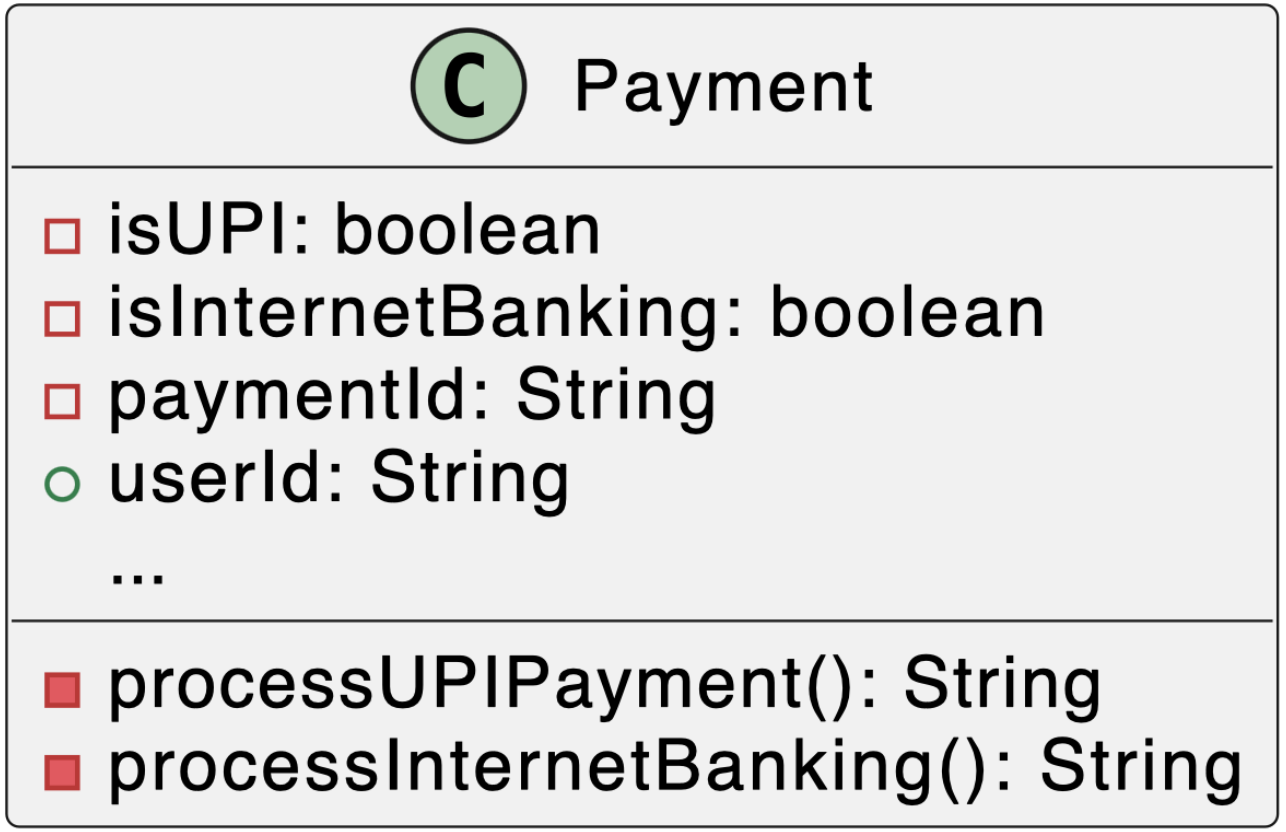
-- Lehmans' Law of Increasing Complexity

Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle". *Journal of Systems and Software*. **1**: 213–221. doi:10.1016/0164-1212(79)90022-0



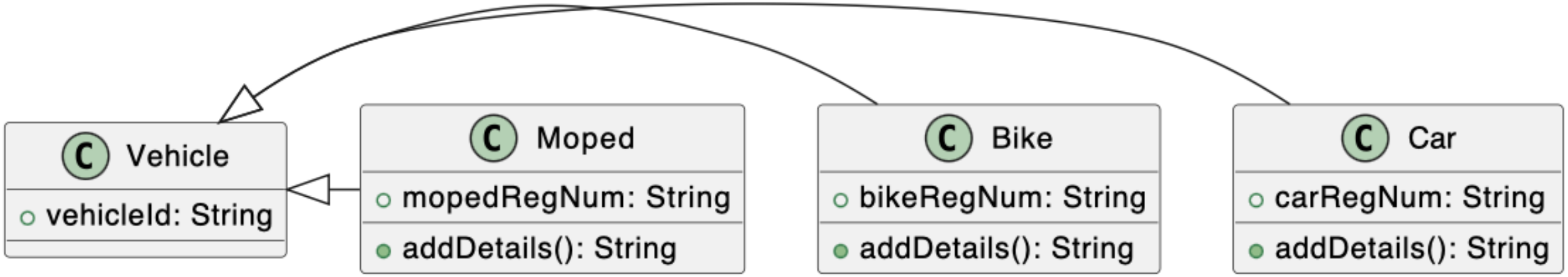
E-bike system is a campus bike renting system where users can rent different types of e-bikes to navigate within and outside the campus. The bikes are placed in dedicated docking station. The users pay before the bike can be unlocked and used. Different payment methods needs to be supported by the app.

Few Examples to Begin with..



Do you see some issues here?

Few Examples to Begin with..



What about this?

Ever heard about Technical Debt?

What is Debt?

debt

/dɛt/

noun

a sum of money that is owed or due.
"I paid off my debts"

Similar: bill account tally

Source: Google dictionary - Oxford, entrepreneur.com

Technical Debt

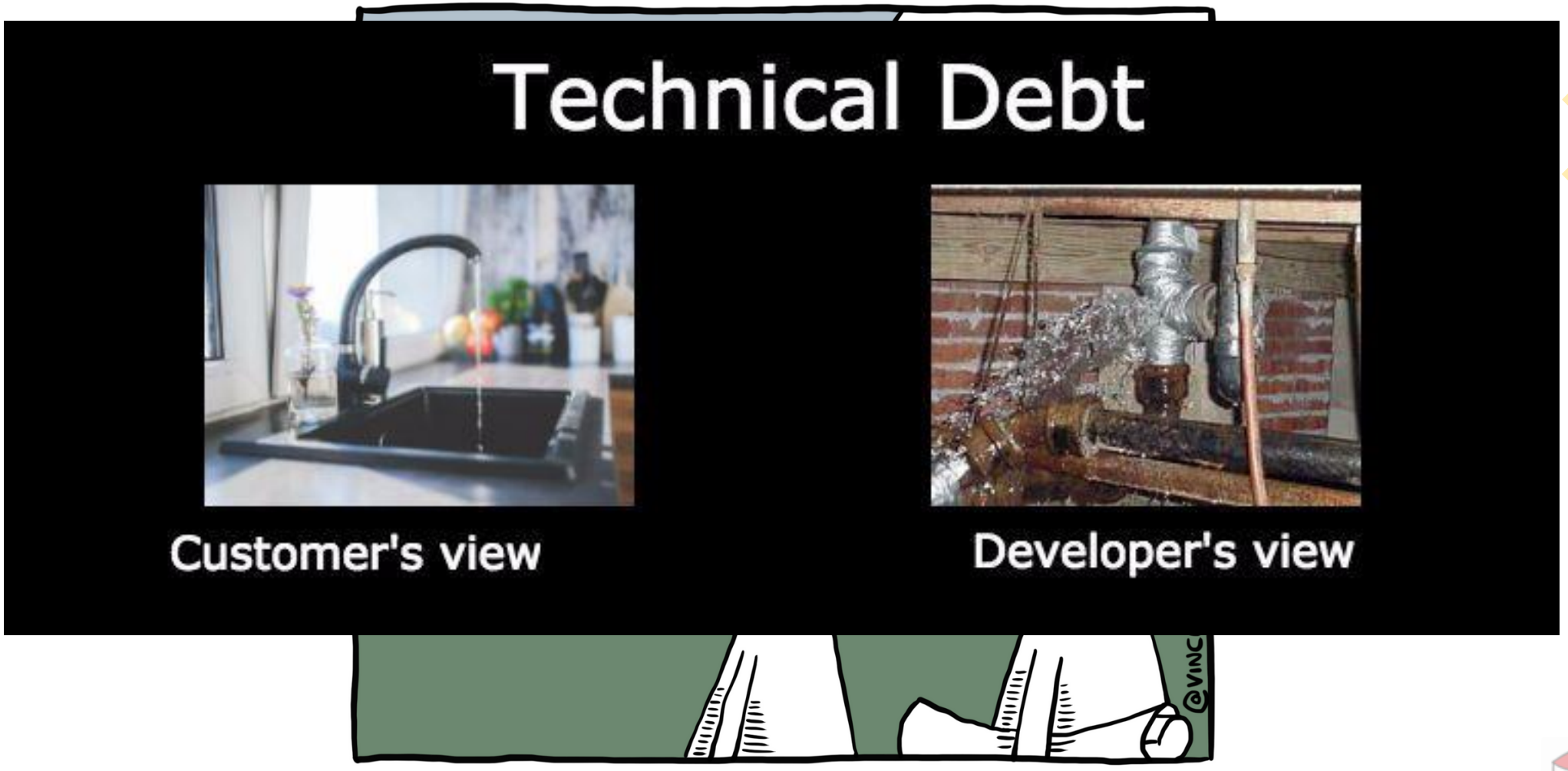
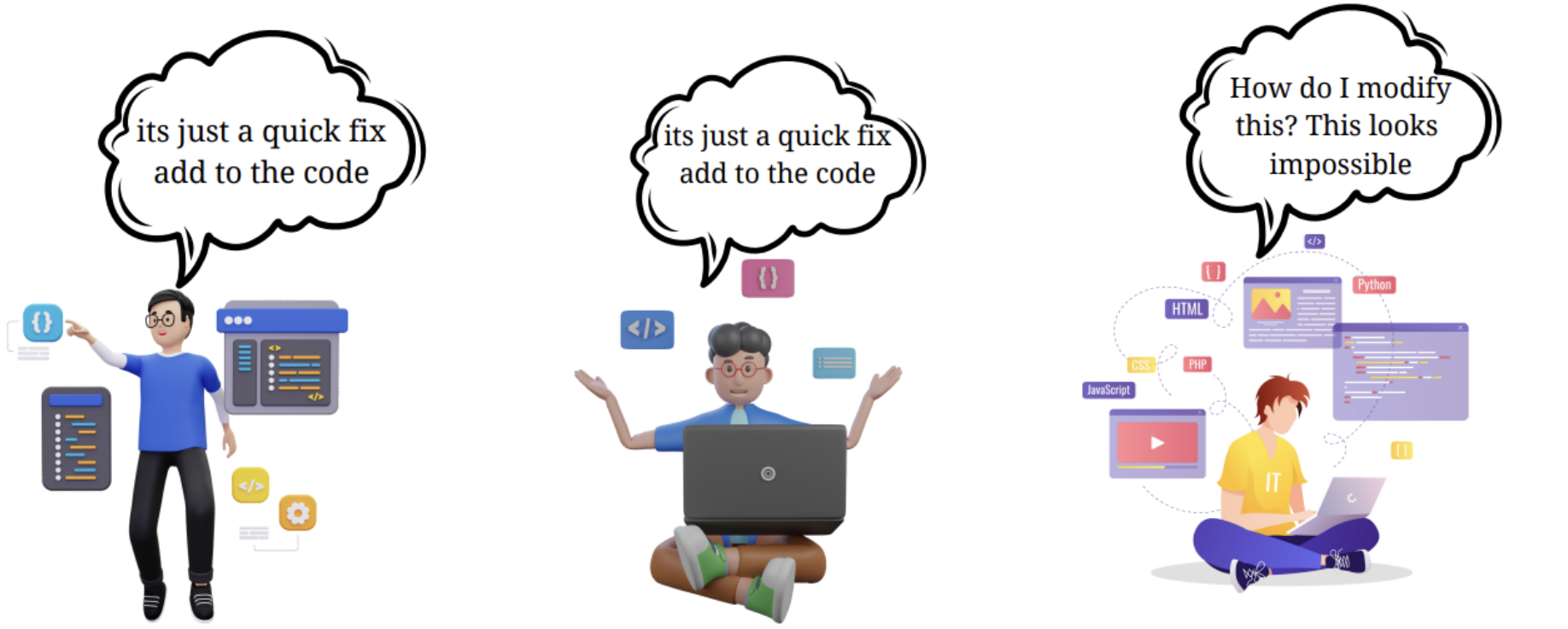


Image source: medium, google images

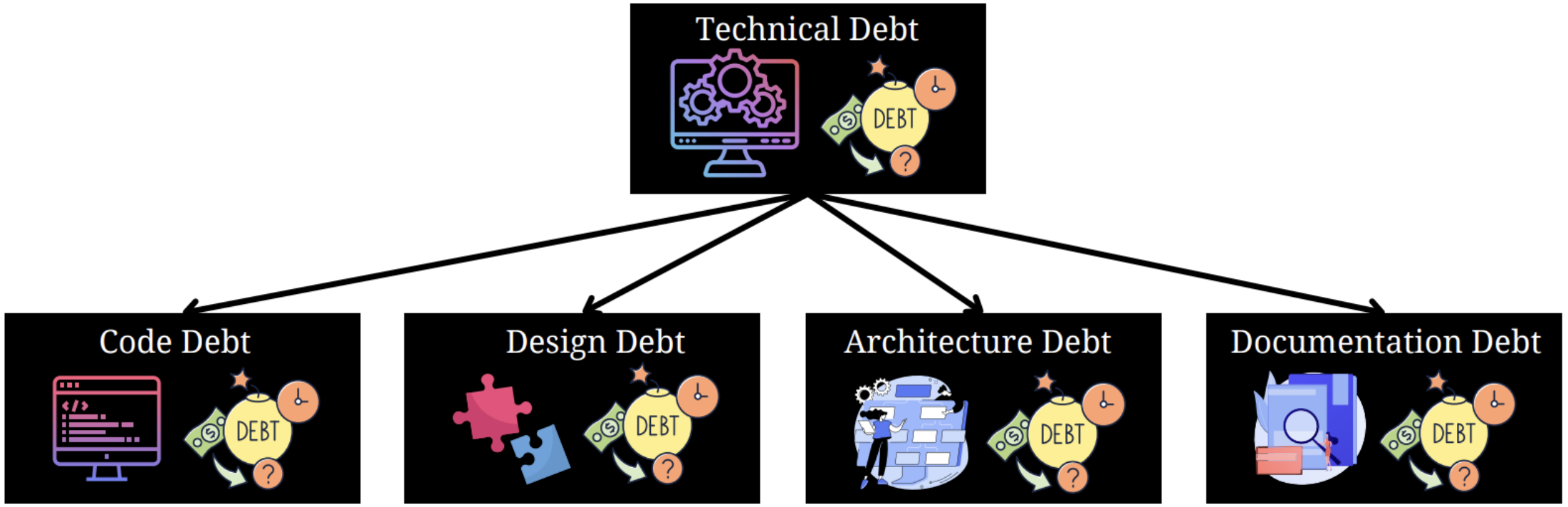
Technical Debt - Definition

Technical debt is the **debt that accrues** when you knowingly or unknowingly make **wrong or non-optimal design decisions**

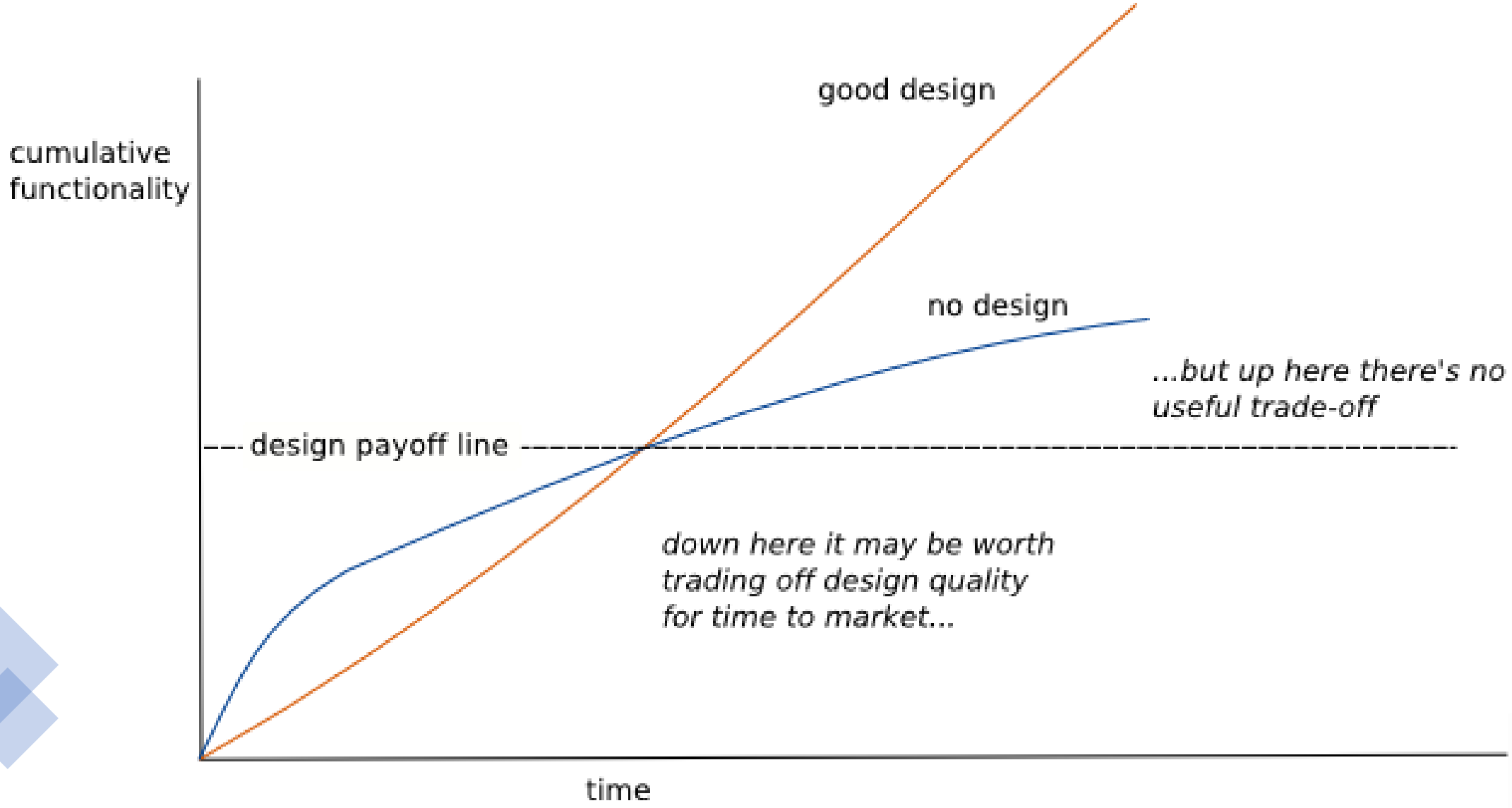
Metaphor coined by *Ward Cunningham*, 1992



Types of Technical Debt



Design Stamina Hypothesis



source: martinfowler.com

Impact of Technical Debt

"One large North American bank learned that its more than 1,000 systems and applications together generated over $2 billion in tech-debt costs" - McKinsey

- Interest is very much compounding in nature – changes has to be done on already existing debt
- Cost of Change becomes extremely high!
- Affects morale of development team
- Huge impact on progress of the business – product and feature delays
- Often considered as the digital dark matter!

Source: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/demystifying-digital-dark-matter-a-new-standard-to-tame-technical-debt

Impact of Technical Debt – An Example Scenario

A successful company in the maritime equipment industry successfully evolved its products for 16 years, in the process amassing 3 million lines of code. Over these 16 years, the company launched many different products, all under warranty or maintenance contracts; new technologies evolved; staff turned over; and new competitors entered the industry.

The company's products were hard to evolve. Small changes or additions led to large amounts of work in regression testing with the existing products, and much of the testing had to be done manually, over several days per release. Small changes often broke the code, for reasons unsuspected by the new members of the development team, because many of the design and program choices were not documented.

What were some things they could have done right?

Source: Ozkaya, I. and Nord, R. . 2019: Managing the Consequences of Technical Debt: 5 Stories from the Field. Carnegie Mellon University's Software Engineering Institute Blog,. https://doi.org/None (Accessed January 13, 2023)

Impact of Technical Debt – Another Case

Southwest Airlines: 'Shameful' Technical Debt Bites Back

BY: RICHI JENNINGS ON JANUARY 5, 2023 — 0 COMMENTS

Welcome to *The Long View*—where we peruse the news of the week and strip it to the essentials. Let's work out **what really matters**.

20 Years of Neglect Led to 'Meltdown'

Last month's débâcle of canceled flights was caused by decades of technical debt. That's the analysis of Columbia University professor **Zeynep Tufekci**.

Analysis: SWA needs a cloud burst

Although there were several contributing factors, a lack of scalability in a critical crew scheduling system led to days of near-total paralysis: In many cases, the staff were in the right place to fly and crew the planes, but the *SkySolver* system had no way of knowing that. Making things worse, manual fallbacks collapsed under the **weight of the workload**.

The answer:…employee scheduling software that debuted around the same time as the Xbox 360 and PlayStation 3. … Southwest pilots have reportedly begged company executives to update the "antiquated" systems since at least 2015.

Eventually someone has to pay for the debt!!

Source: https://devops.com/southwest-technical-debt-richixbw/

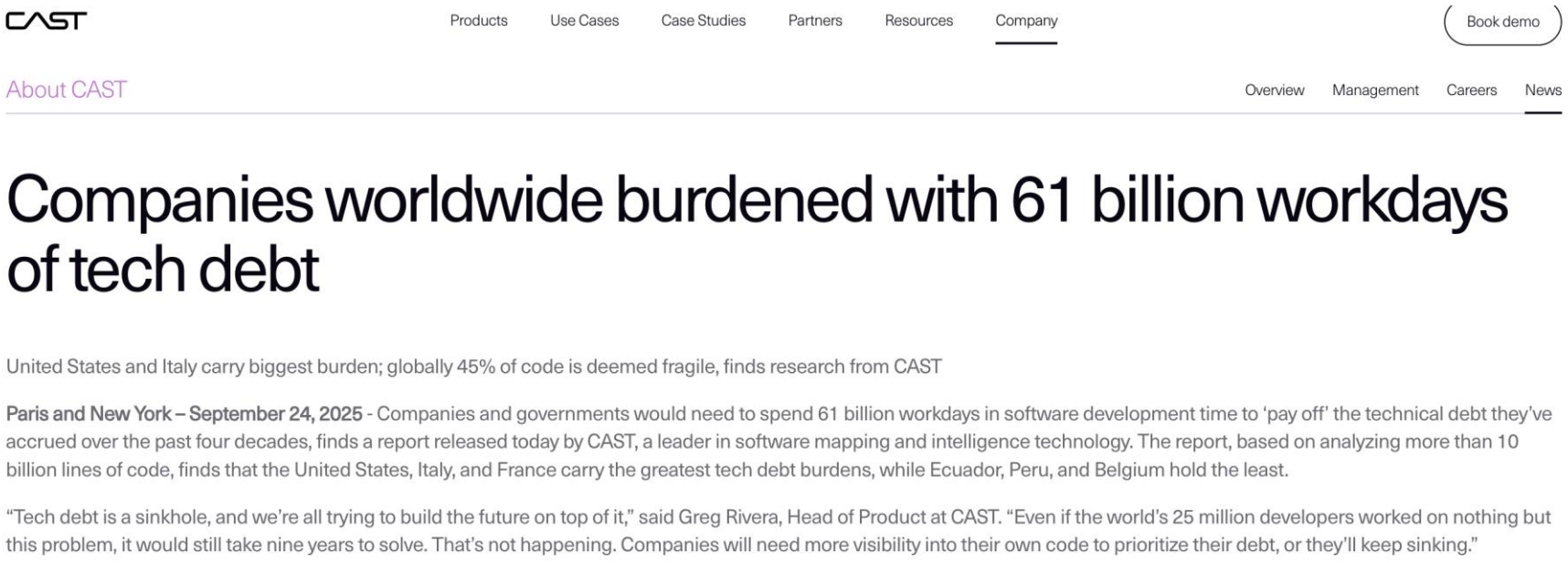
Doesn’t stop there

Technical debt India's key tech challenge towards $1 trillion digital economy goal by 2030: CAST CEO

Technical debt, defined as the accumulation of outdated or inefficient code, imposes a global cost of over $1.3 trillion annually, according to McKinsey, and India is no exception.

IANS

Updated On Jan 28, 2025 at 08:13 AM IST



Source: Economic Times

Reasons for Technical Debt

Everyone in the decision making could be blamed – Architects, developers, managers.. but that doesn't end there. There are many other reasons..

- Schedule pressure – Copy paste programming
  - Its not always about getting the syntax right and making something work
- Lack of skilled designers – Poor applications of design principles
  - Lack of awareness about best practices
  - Leading in the wrong direction
- Lack of awareness of key indicators and checks - Design issues
  - Periodic review of design and making changes can go a long way!!

Lot of research being done!

Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips
{dsculley,gholt,dgg,edavydov,toddphillips}@google.com
Google, Inc.

Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison
{ebner,vchaudhary,mwyoung,jfcrespo,dennison}@google.com
Google, Inc.

Abstract

Machine learning offers a fantastically powerful toolkit for building useful complex prediction systems quickly. This paper argues it is dangerous to think of these quick wins as coming for free. Using the software engineering framework of *technical debt*, we find it is common to incur massive ongoing maintenance costs in real-world ML systems. We explore several ML-specific risk factors to account for in system design. These include boundary erosion, entanglement, hidden feedback loops, undeclared consumers, data dependencies, configuration issues, changes in the external world, and a variety of system-level anti-patterns.

ICSME 2026
Benevento, Italy

42nd IEEE International Conference on Software Maintenance and Evolution

Mon 14 - Fri 18 September 2026

Source: respective conference websites

Opportunities for Startup!

Ardoq

Platform Customers Partners Resources Request a Demo

Technical Debt Management

Win the Battle! Tackle technical debt with Ardoq

Get Started With Ardoq

Lot of space!!

Arcan

Product About Resources BOOK A DEMO

Discover the hidden costs of your architecture

Arcan cleverly supports you in monitoring the evolution of Architectural Technical Debt, understanding its causes and in preventing its accumulation.

Technical debt principal estimation

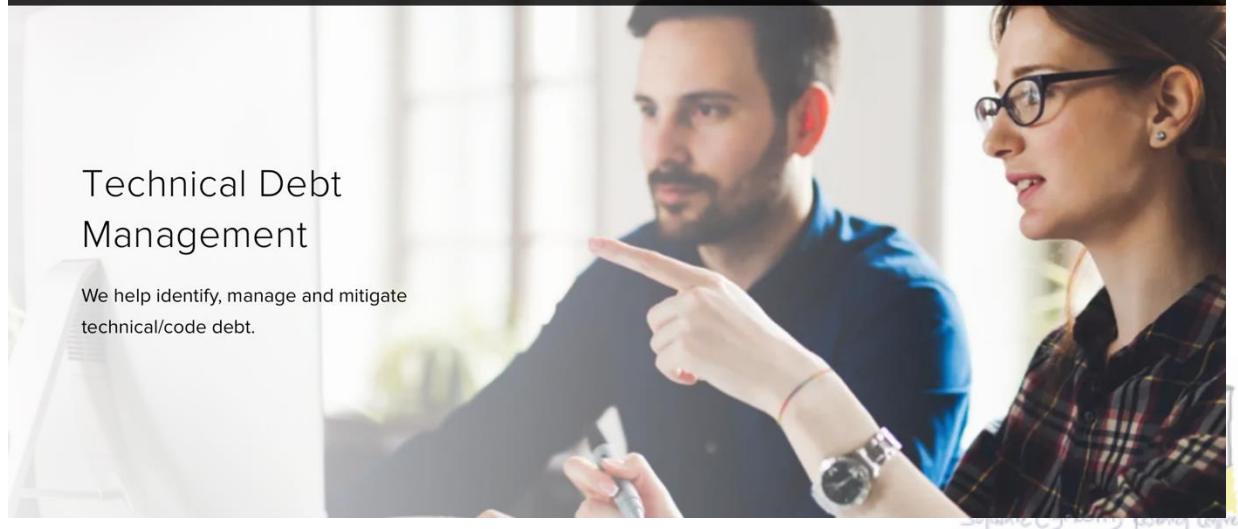
A condensed indicator of the current debt of your architecture, in terms of cost (in € or $) necessary to achieve the optimal state, available directly on the Arcan dashboard.

Technical Debt evolution monitoring

Connect your Git repository to visualize the evolution of technical debt principal over time and consult the analysis results for each version.

10Pearls

Company Services Platforms Industries Insights Careers



Managing Technical Debt

- Increase awareness about tech debt
  - Being aware is the best start
  - Create goals keeping this in mind
- Detect and repay tech debt systematically
  - Identify instances of debt (huge impact)
  - Create systematic plan on recovery
- Prevent accumulation of tech debt
  - Once under control, prevent further accumulation
  - Perform regular monitoring
- Companies should allocate some budget for tech debt

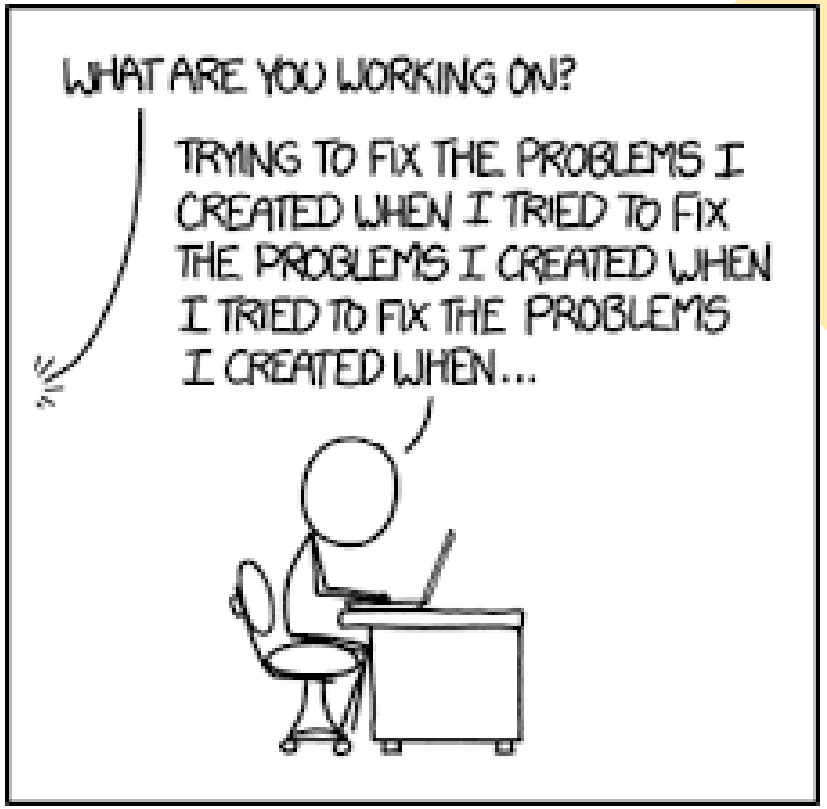


Image source: xkcd

Key Major Questions

1. Why do even good developers write bad software?
2. How do we fix our software?
3. How to know if the software is “bad” even when its working fine?

Refactoring!

“Any fool can write code that a computer can understand. Good Programmers write code that humans can understand”

Martin Fowler
Thoughtworks

Image source: thoughtworks

What is Refactoring?

It is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour

-- Martin Fowler

Image source: imageflip.om

What is Refactoring?

- Refactoring is not always a clean up of code!
- Goal is to make software easier to understand and modify
- Think of performance optimization
- Refactoring does not or should not change behavior – No change to external user [Changing hats]
- Not always same as:
  - Adding features
  - Debugging code
  - Rewriting code

Image source: imageflip.com

When to Refactor?

- Follow the rule of three
  - First time, just get it done
  - Second time to do something similar, duplicate
  - Third time, just refactor
- Refactor when you add a function (feature)
  - When adding new feature, make it more effective and efficient
- Refactor when you fix a bug
  - Bug by themselves can be good indicators – Are they becoming more common?
- Refactor when you do code reviews
  - Create review groups for code reviews, new perspective may lead to refactoring

Image source: imageflip.com

Some Common Refactoring – Low Level refactoring

- IDEs provide a lot of support
- Variable/method/class renaming
- Extraction of duplicate code snippets
- Change in method signature
- Method or constant extraction
- Warnings about unused variables, parameter uses/declarations
- Auto-completion support and minimal documentation support

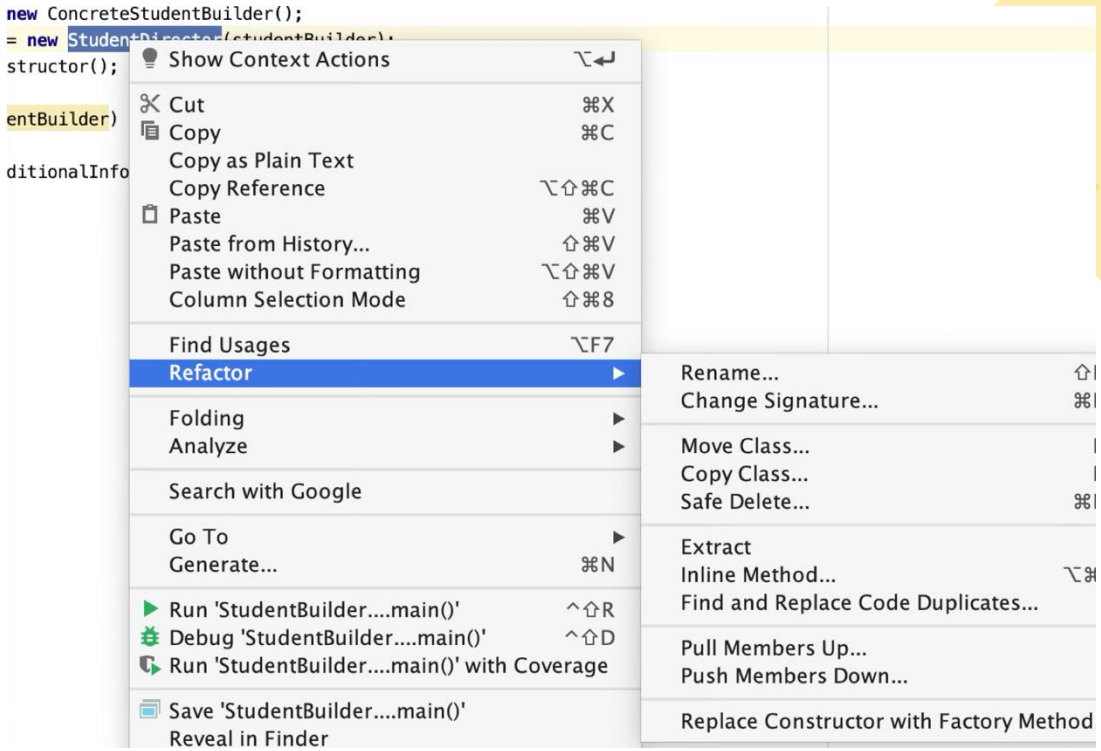


Image source: Intellij.com

High-level refactoring - Challenges

- Much more complex – has dependency on use case, context
- Risk of introducing bugs – Changes in design can introduce new issues
- Testing can become difficult – New test cases needs to be added, overall

Behavior may change [ideally not!]

- Communication of changes – Changes can be more abstract and harder to explain
- Measuring the impact – Changes can be harder to quantify

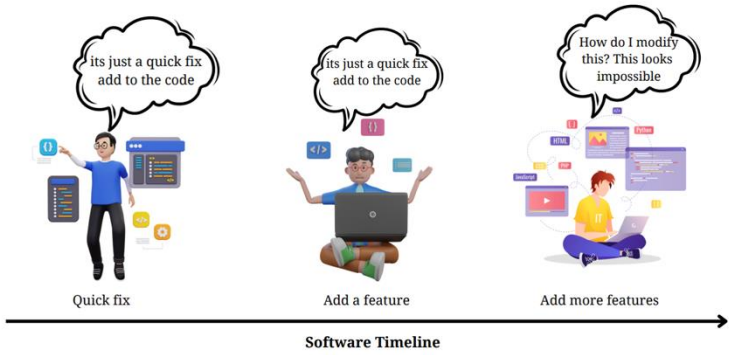
Image source: imageflip.om

Summary So Far

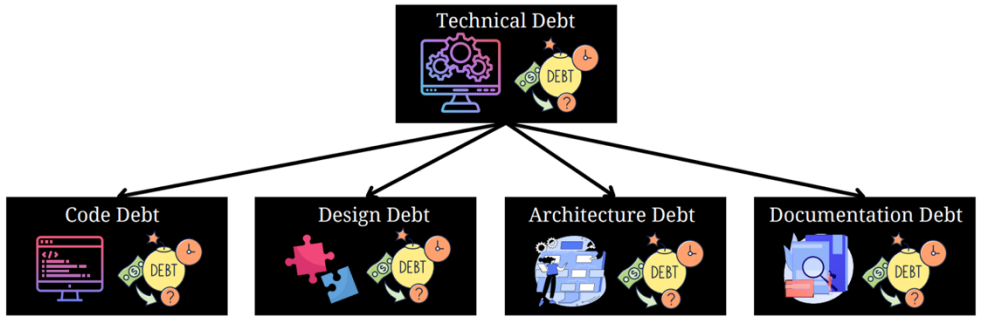
Technical Debt - Definition

Technical debt is the **debt that accrues** when you knowingly or unknowingly make **wrong or non-optimal design decisions**

Metaphor coined by *Ward Cunningham*, 1992



Types of Technical Debt



What is Refactoring?

It is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour

-- Martin Fowler

Image source: imageflip.om

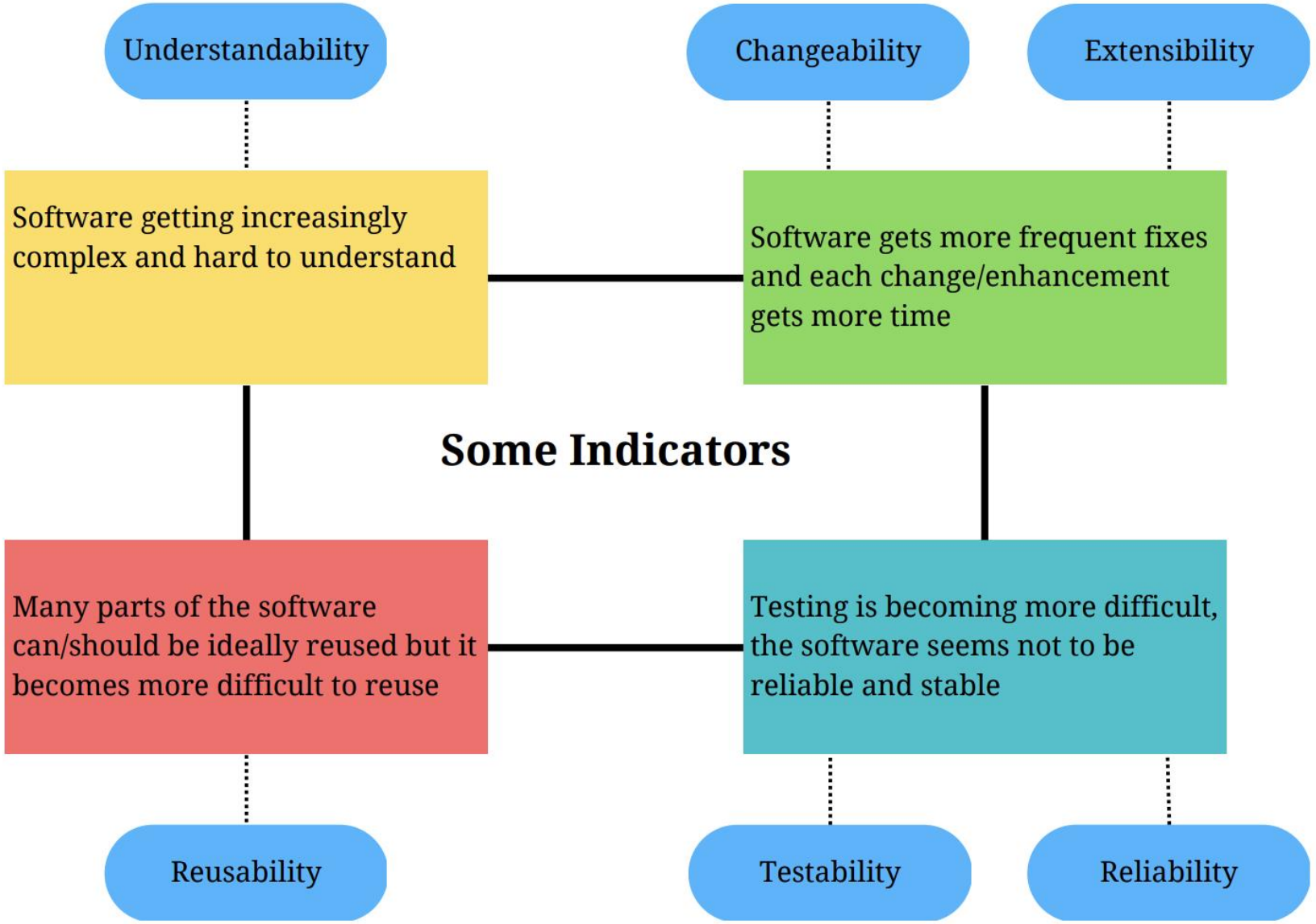
High-level refactoring - Challenges

- Much more complex – has dependency on use case, context
- Risk of introducing bugs – Changes in design can introduce new issues
- Testing can become difficult – New test cases needs to be added, overall Behavior may change [ideally not!]
- Communication of changes – Changes can be more abstract and harder to explain
- Measuring the impact – Changes can be harder to quantify

Image source: imageflip.om

How to Identify Technical Debts and Refactor?

Software Quality as an Indicator

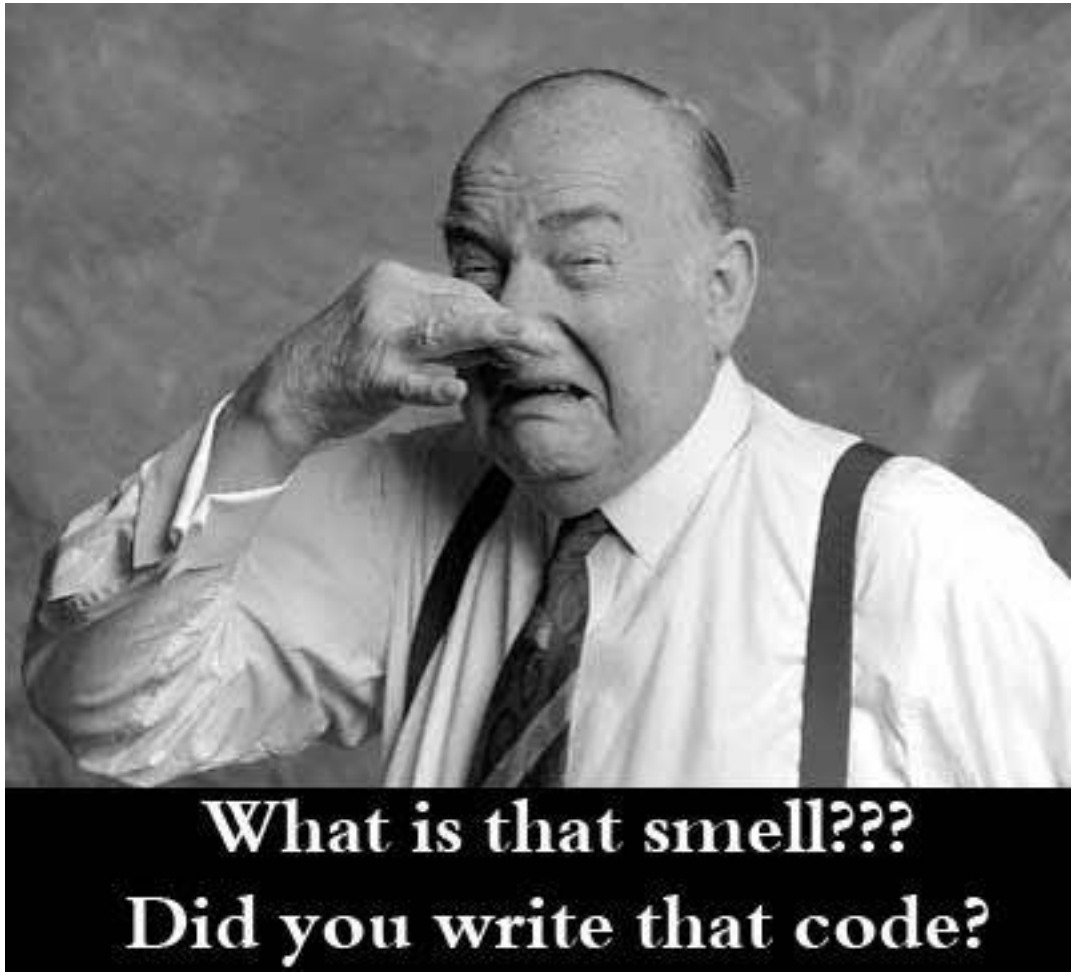


How to Refactor?

- Identify the refactoring points
- Create a refactoring plan
- Make a backup of the existing codebase: Versioning system
- Use semi-automated approach: Some tool support is always available
- Perform the refactoring
- Test if everything works like before! – Test extensively (new bugs, broken functionalities, etc.)
- Repeat the process

Remember: Refactoring is not just a one time activity!!

Code Smells? You heard that right!



Source: https://engineering.99x.io/natural-odor-of-a-developer-f7f7fbb838e3

Refactoring Points - Things starts to rot and **Smell**

Code Smells and so does design – You heard that right!!!

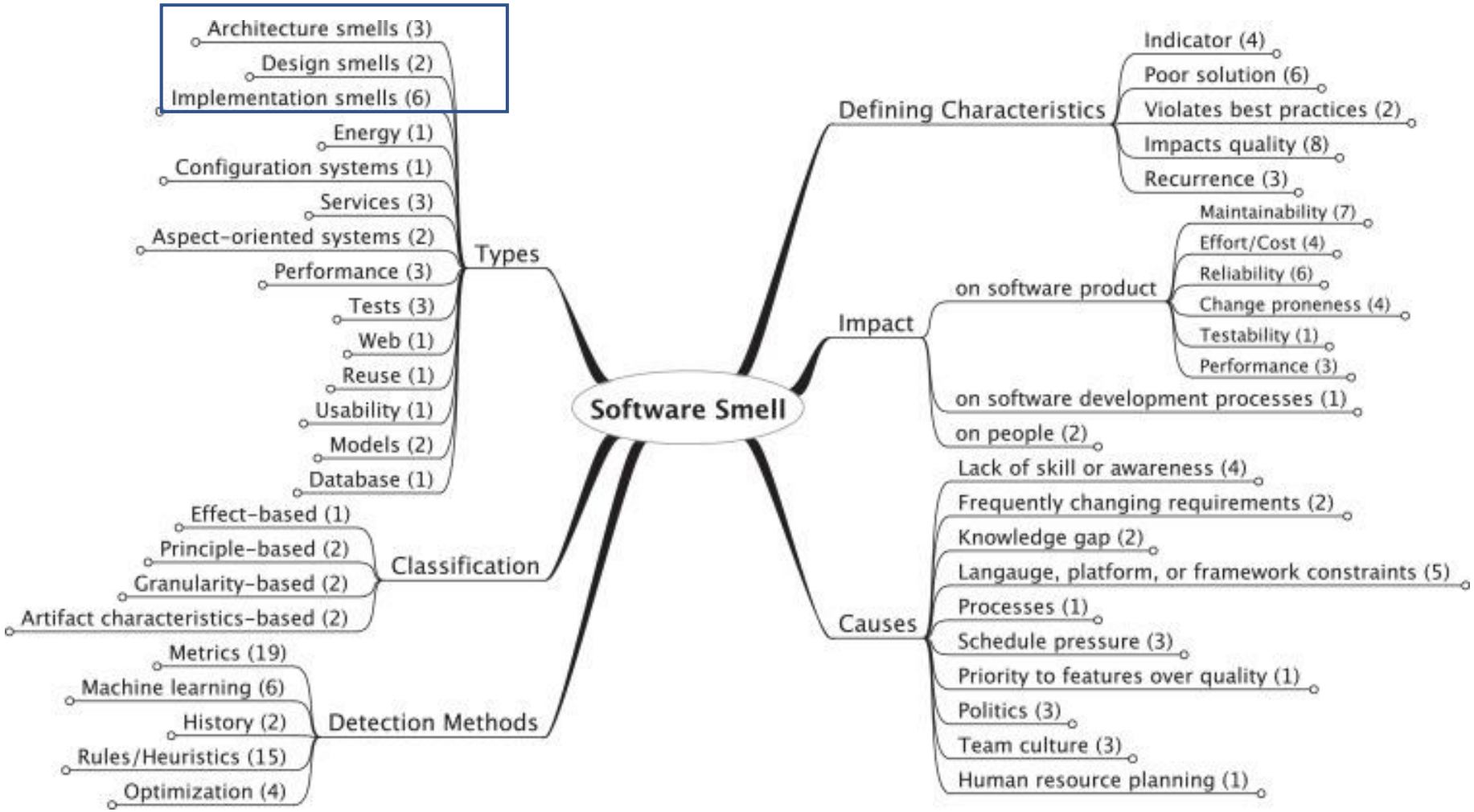
”smell”, Coined by Kent Beck in 1999

Smells are certain structures in the code that **suggest** (sometimes they scream for) the **possibility of refactoring**

A ”bad smell" describes a situation where there are hints that suggest there can be a **design problem**

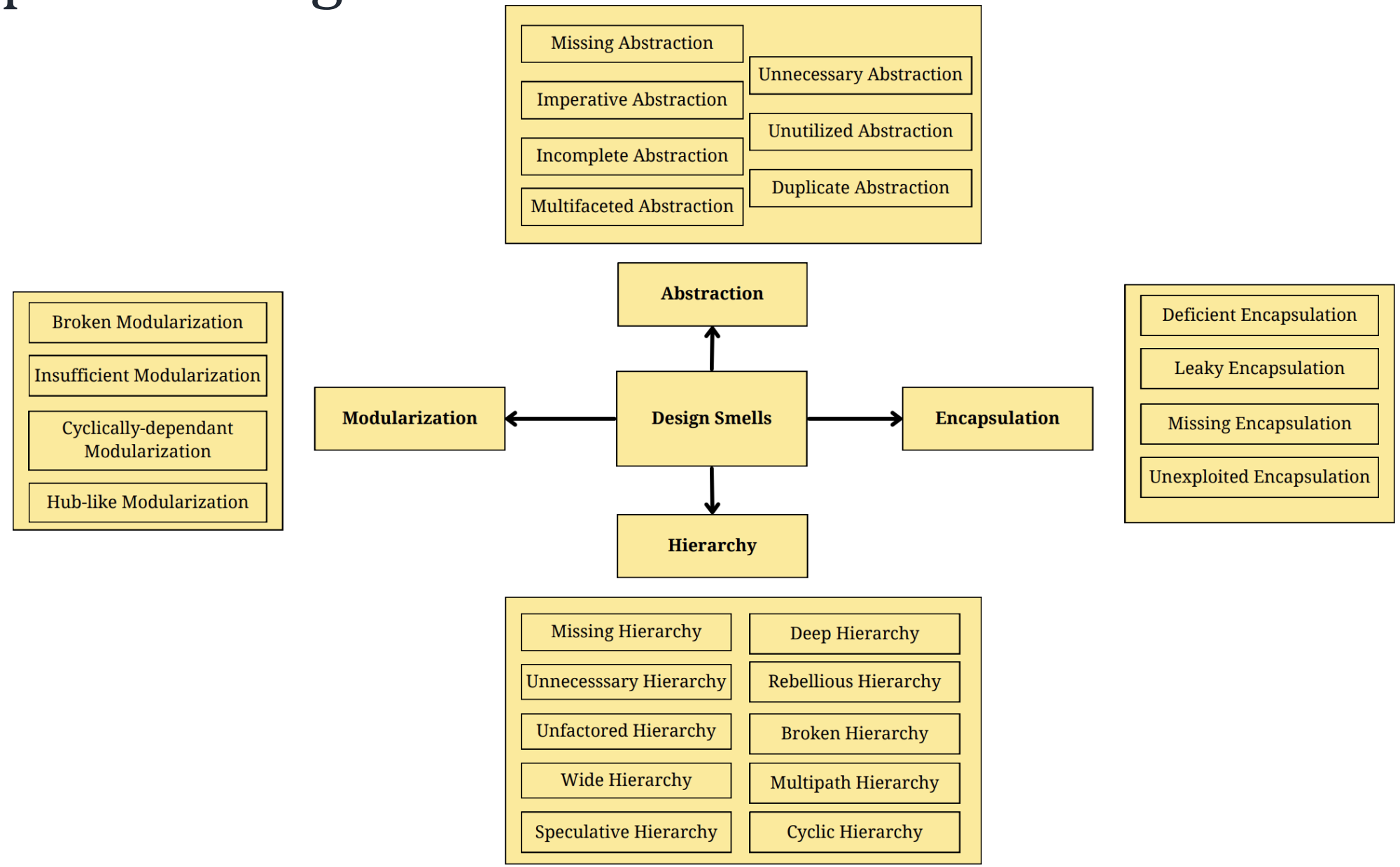
Many different definitions - *https://zenodo.org/record/1066135#.Y8PcXS8RpQI*

Many methods, reasons, ways to detect..



Tushar Sharma, Diomidis Spinellis, **A survey on software smells**, Journal of Systems and Software, Volume 138, 2018

Types of Design Smells



Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com
Twitter: @karthi_ishere