Antonio Martini
Professor in Software Engineering
University of Oslo

Course IN5140
2023-09-27

MANAGING TECHNICAL DEBT

Agenda

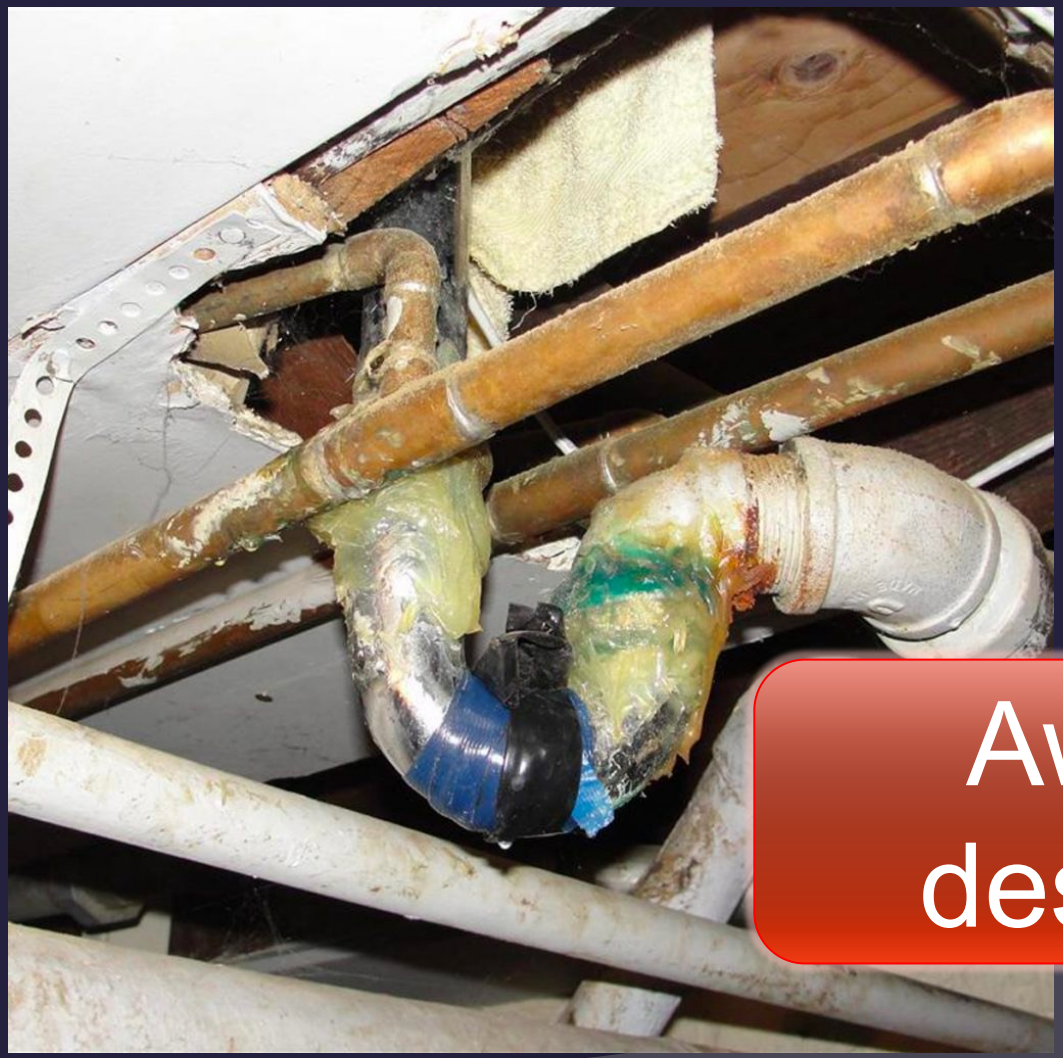
- Recap
  - What is Technical Debt?
  - Why and how do we manage Technical Debt?
- Managing Technical Debt in Agile
  - Several aspects
- Managing Technical Debt
  - Guest lecture from Visma (Mili)
- Managing Security Debt
  - Guest lecture from Visma (Maren)

Recap

Technical Debt

What the users see

What the developers see



Awful design!

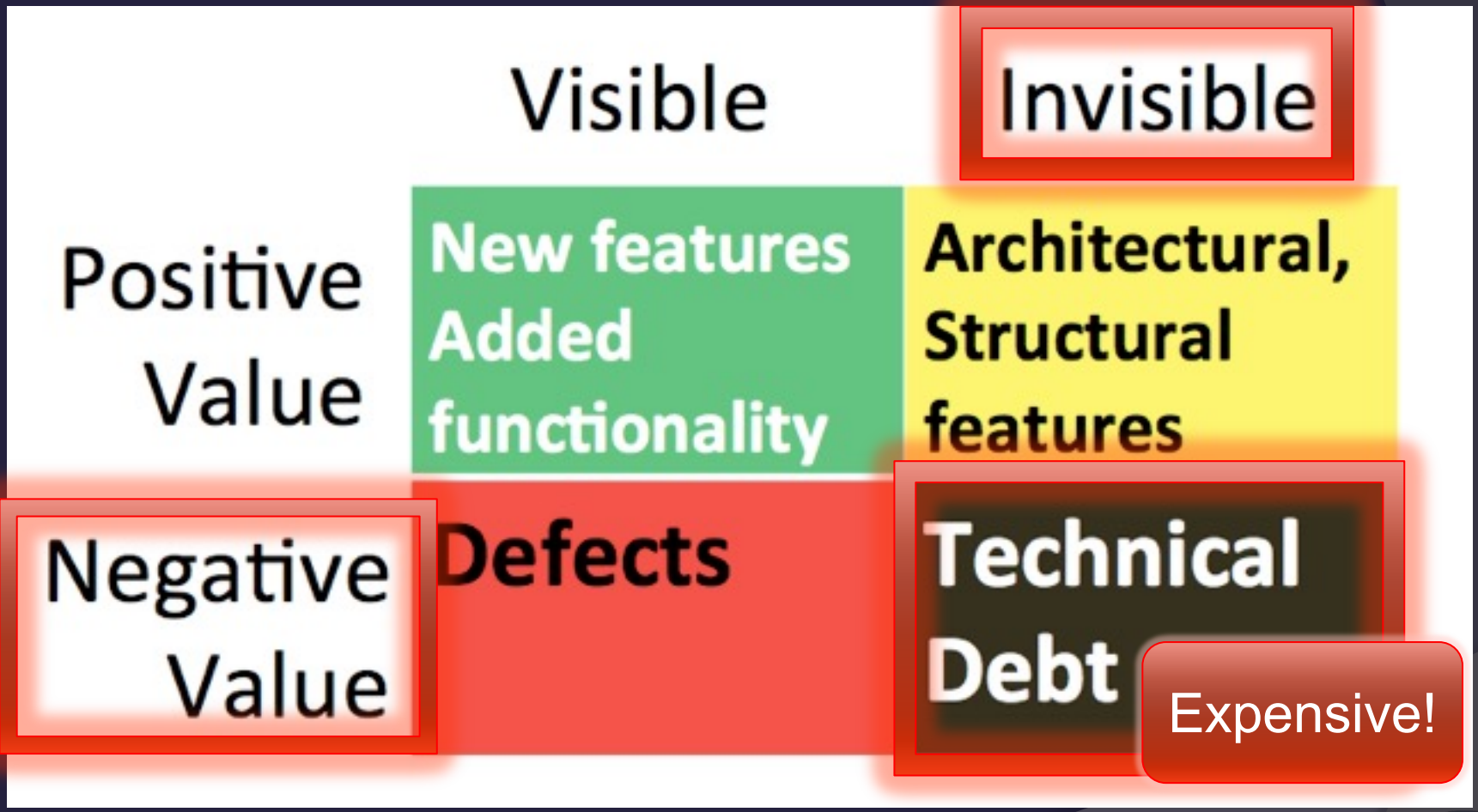
Technical Debt in a nutshell

Current Definition

- *In software-intensive systems, technical debt is a design or implementation construct that is expedient in the short term, but sets up a technical context that can make a future change more costly or impossible. Technical debt is a contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability*

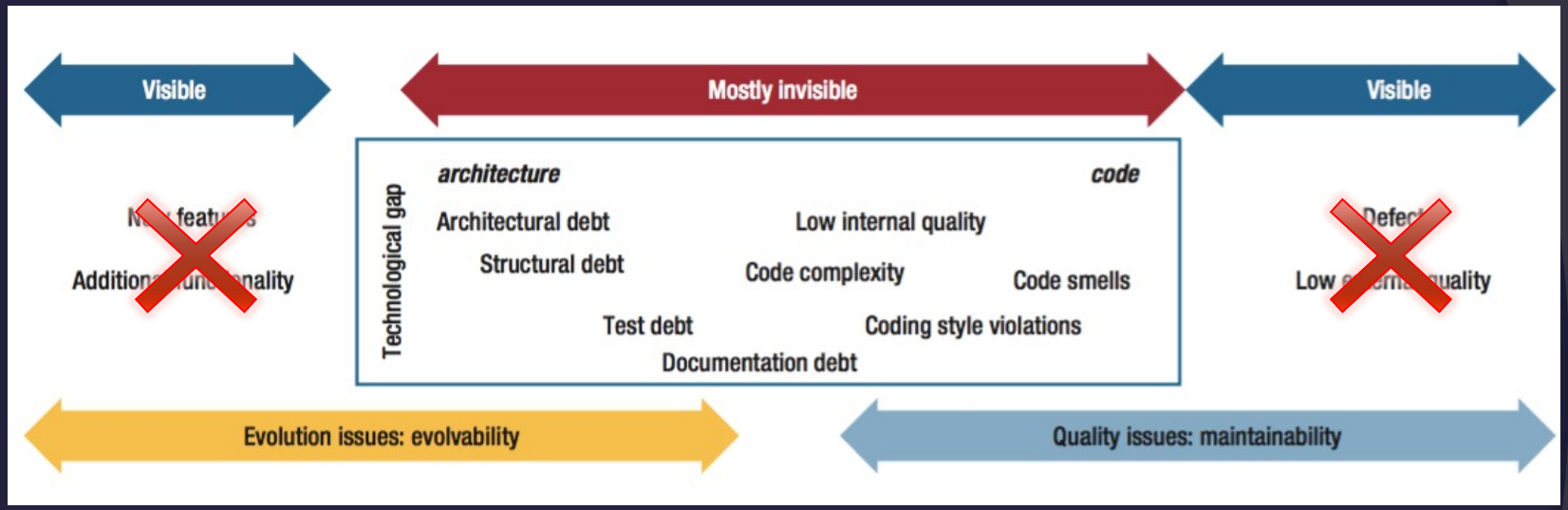
P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)

First of all: What is Technical Debt?



P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*

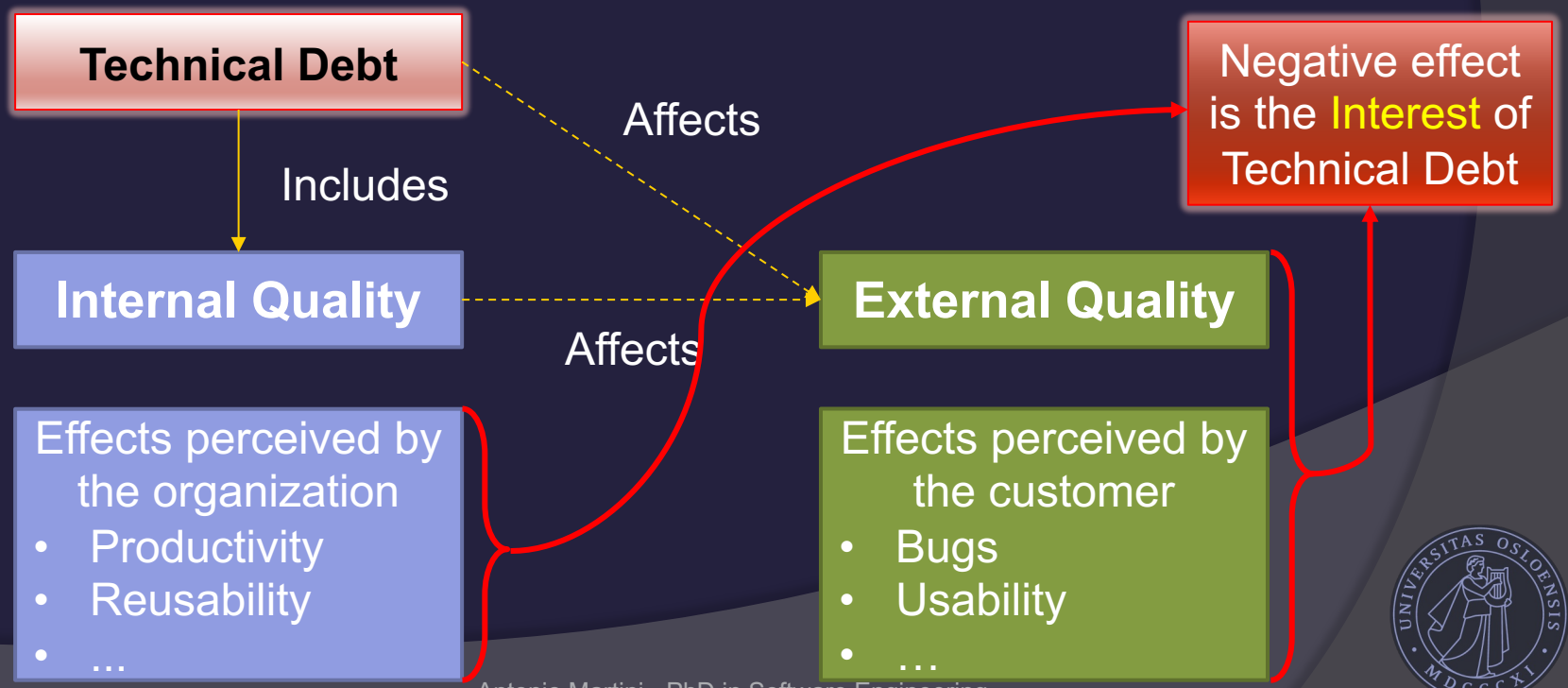
The TD landscape of kinds of TD



P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” *IEEE Software*

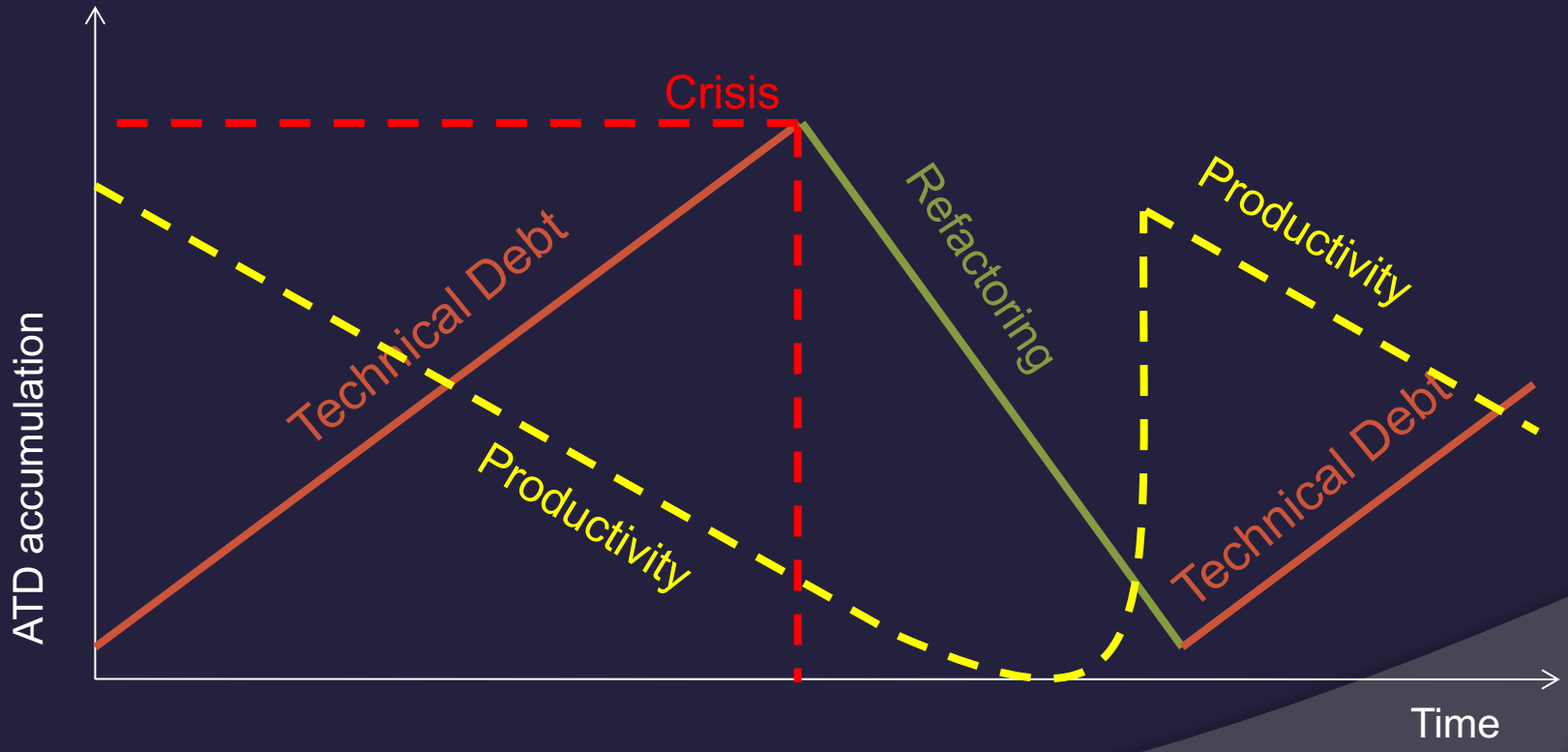
What is technical debt in practice?

- TD includes internal quality issues, not external quality
  - TD is not a bug!
- External quality *might* be influenced by internal quality
  - Example: it might be more difficult to fix a bug *because* of the technical debt



So, what happens in the end?

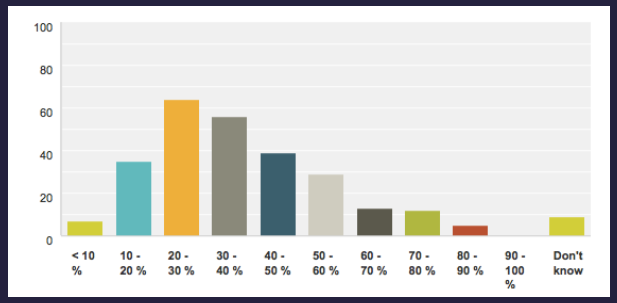
- Research study in 7 organizations *
- The accumulation of Technical Debt leads to development crises



* Martini, A., Bosch, J., Chaudron, M., 2015. “Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study,” *Information and Software Technology*.

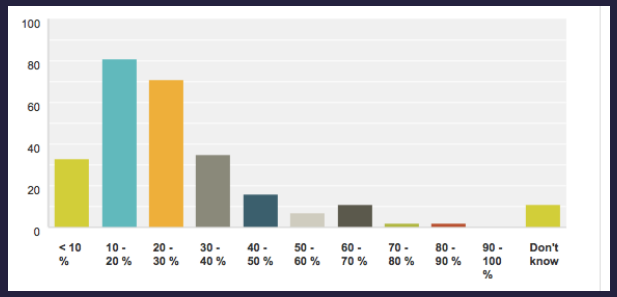
How much is the interest?

- 269 respondents in 15 organizations
- 40 developers tracked over 3 months
  - Estimated Interest (waste) of TD



30% waste

  - Estimated Cost of TD management



25% waste

- Martini et al.: "*Technical Debt Tracking: Current State of Practice*", Journal of Science of Computer Programming, 2018
- T. Besker, A. Martini, and J. Bosch, "The Pricey Bill of Technical Debt - When and by whom will it be paid?," in Proceeding of ICSME 2017, Shanghai, China.

But TD can be **useful** in some cases

- When high business risk is involved
  - E.g. startups
    - High chances of failing
    - Need to save capital
    - They **need** technical debt in the beginning!*
    - But they need to remove it before they **grow**
- Prototyping
  - **Helps** understanding if something is **feasible**
    - Problem: prototypes **should not end up** in production code

* T. Besker, A. Martini, R. Edirisooriya Lokuge, K. Blincoe and J. Bosch, "Embracing Technical Debt, from a Startup Company Perspective," 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, 2018

You cannot avoid TD...

- But TD needs to be managed
- The most risky (with more interest) issues need to be fixed (first)
- Prioritization is key

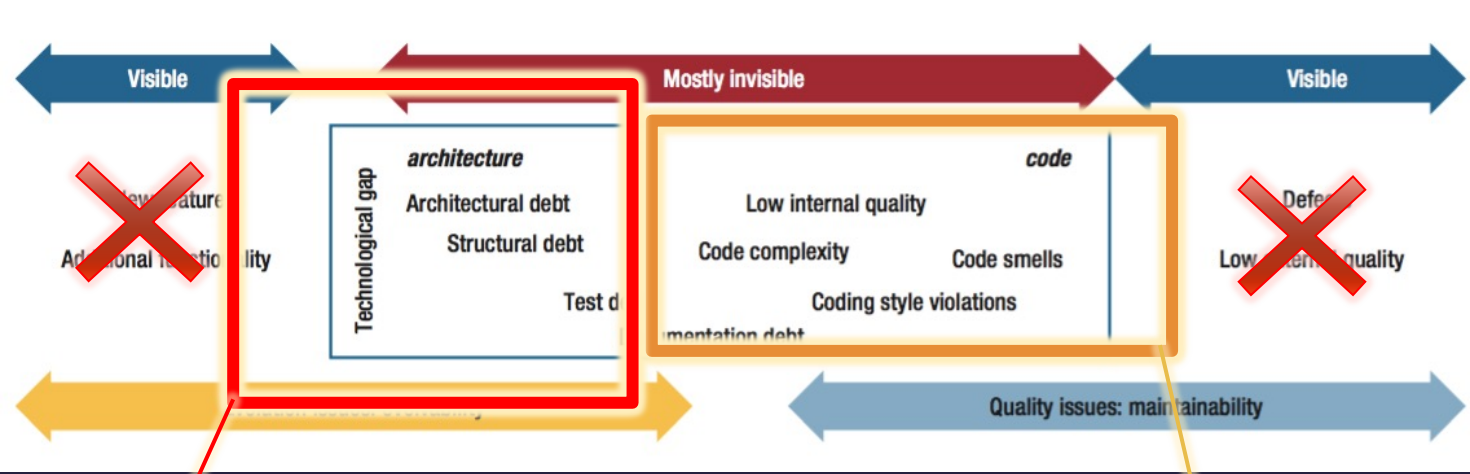
Managing Technical Debt

Identification

- How do we know if we have technical debt?
- The concept of (code) “smell”
  - *a code smell is any characteristic in the source code of a program that possibly indicates a deeper problem* *
- How do we find code smells?

Tufano, Michele; Palomba, Fabio; Bavota, Gabriele; Oliveto, Rocco; Di Penta, Massimiliano; De Lucia, Andrea; Poshyvanyk, Denys "When and Why Your Code Starts to Smell Bad" 37th IEEE International Conference on Software Engineering 2015

Identification of different kinds of TD

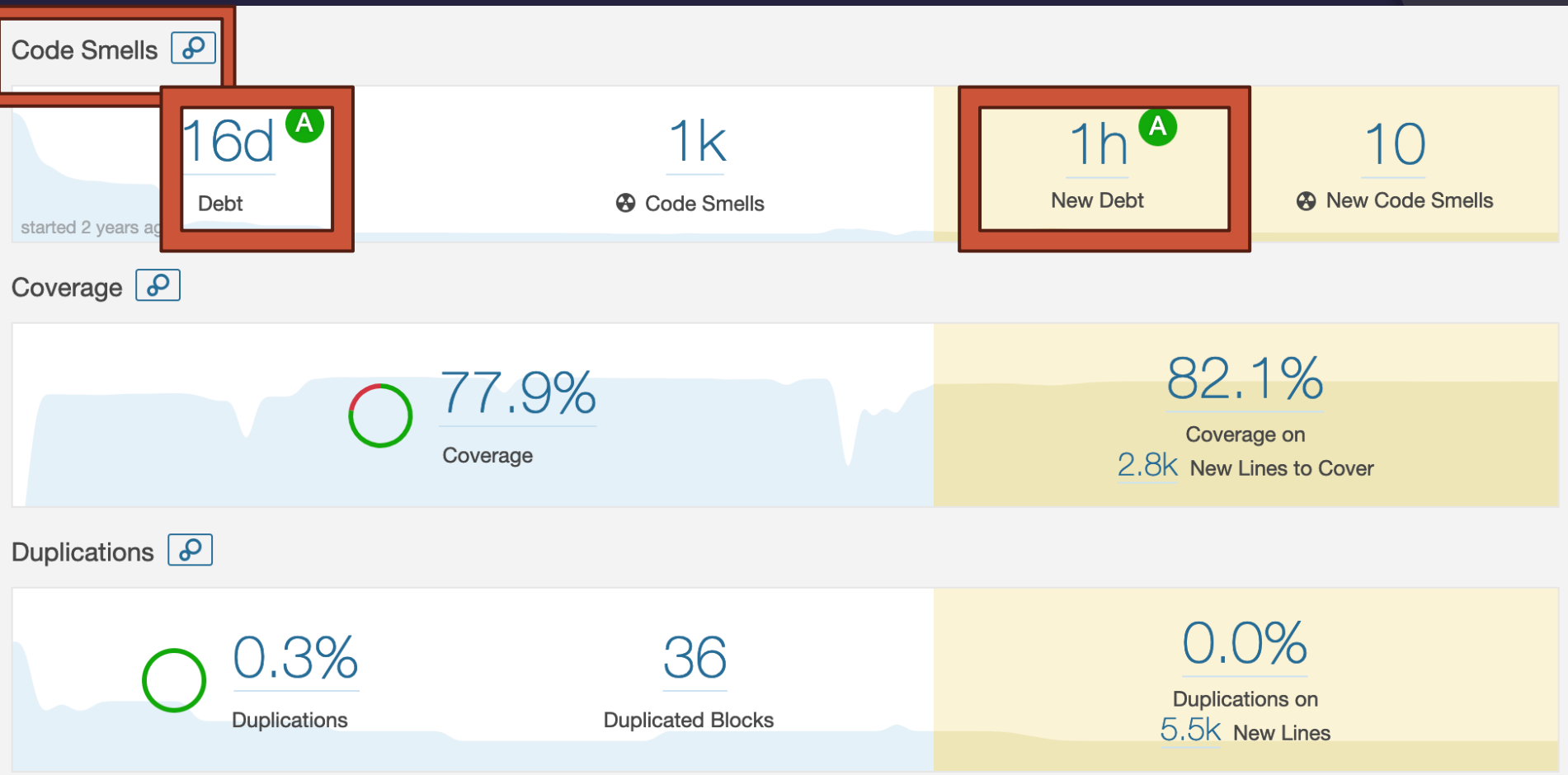


Manual or Invisible

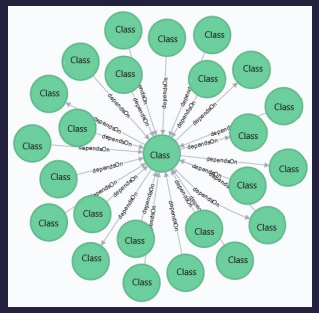
Automatic Tools

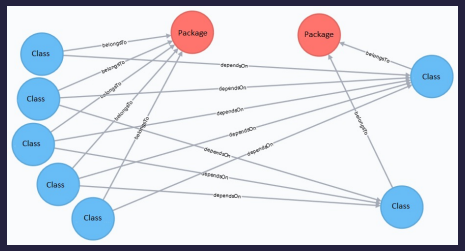
(Do not show impact of Technical Debt)

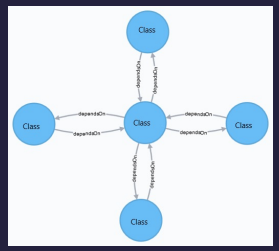
Static code analyzers (e.g. SonarQube)



Architectural smells (e.g. Arcan)







50% increment awareness of Architectural Debt

* Martini, Antonio; Arcelli Fontana, Francesca; Biaggi, Andrea & Roveda, Riccardo (2018). Identifying and Prioritizing Architectural Debt Through Architectural Smells, ECSA 2018

Refactoring

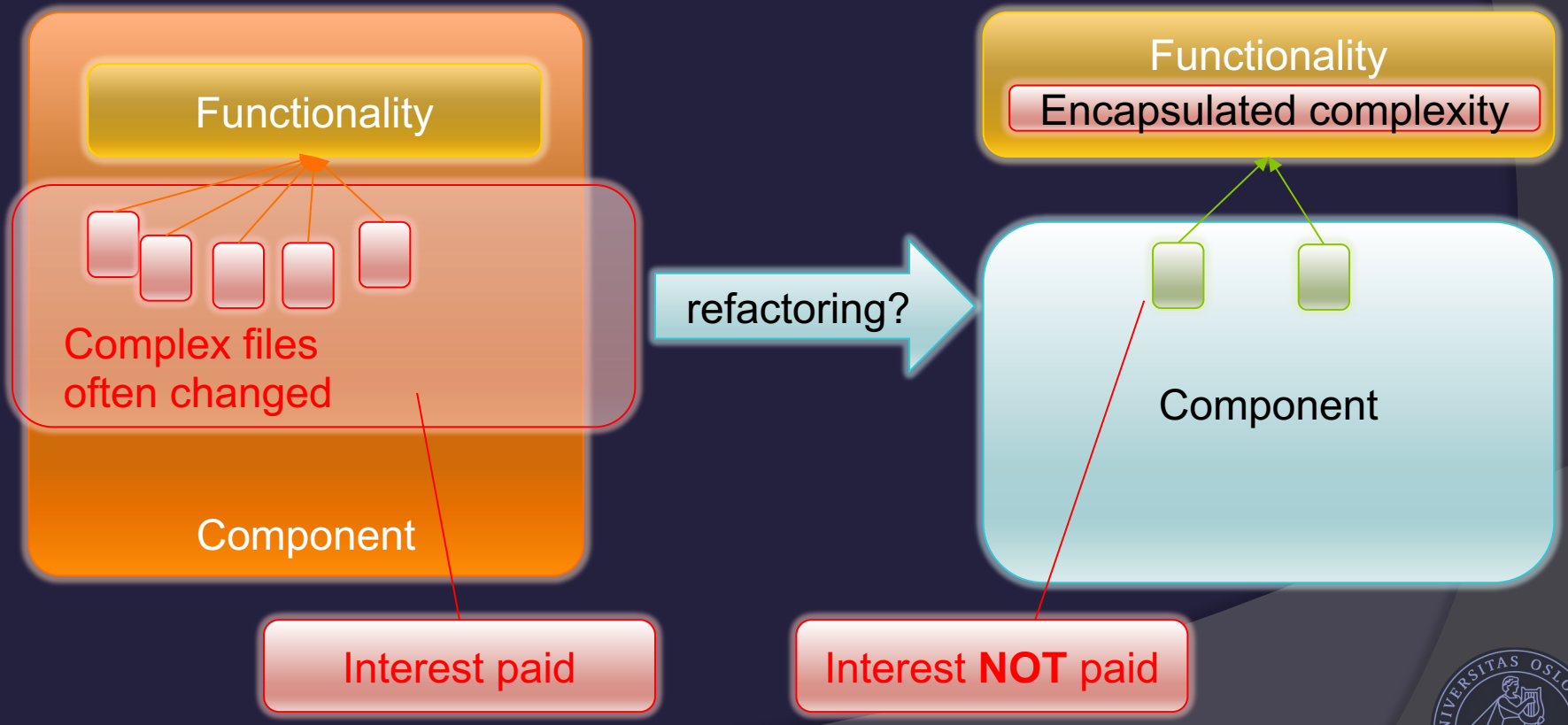
- *restructure (the source code of an application or piece of software) so as to improve operation without altering functionality*

  (Oxford dictionary)

- It's a reactive approach
  - Agile mostly advocate it to compensate when avoiding "big upfront"
  - Most of the time overlooked
  - Large refactorings

Maintainability and complexity

- Complex files need to be decreased and encapsulated



How do large companies manage TD?

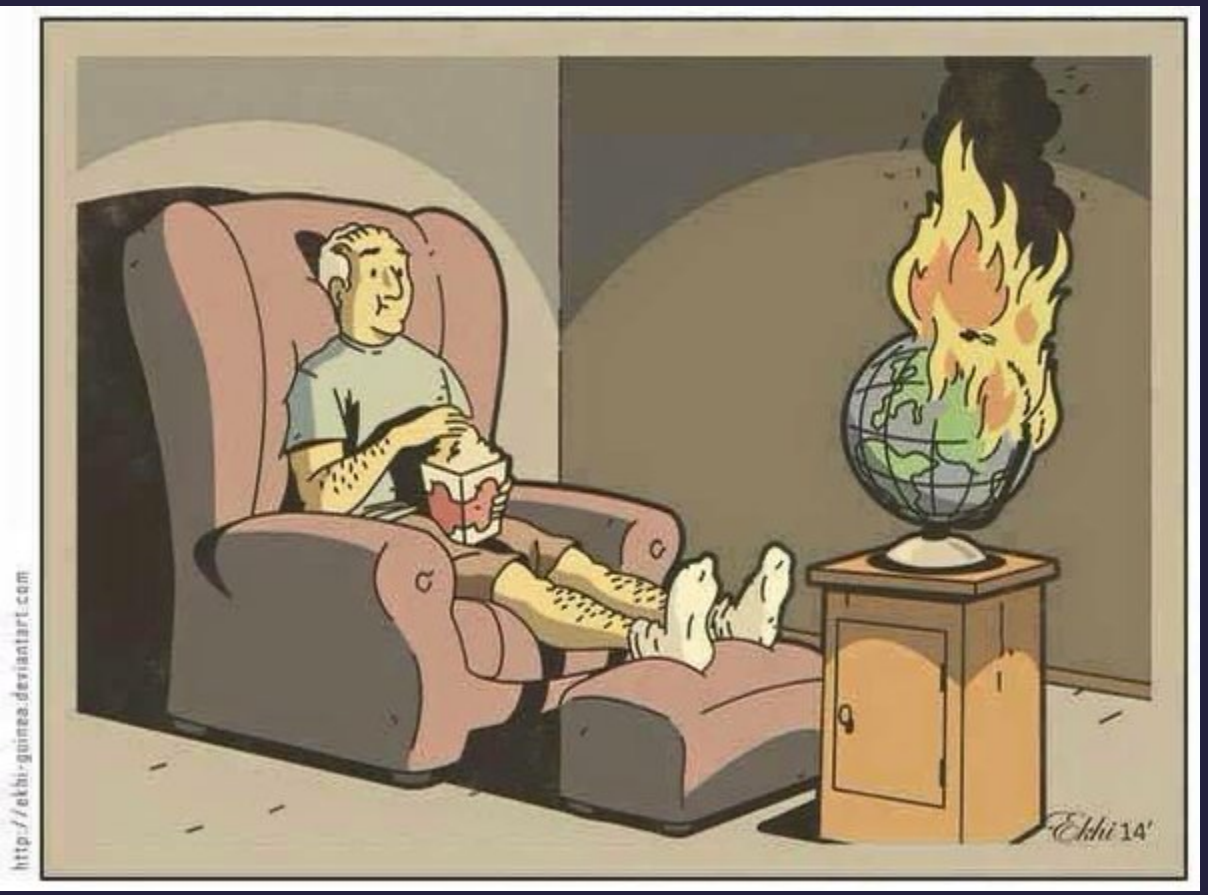
- Companies manage TD with different levels of maturity*

* Martini, Besker, Bosch: "*Introducing Technical Debt Tracking in Large Software Companies*", accepted at APSEC 2016 and in press

Level 0 - Unaware

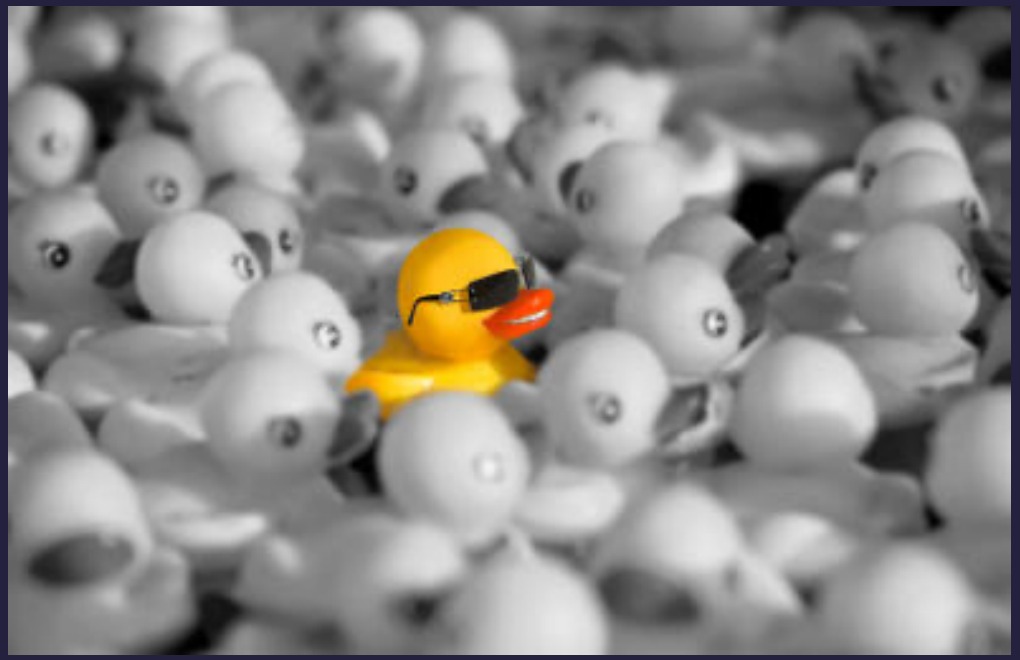
- There is no awareness of TD in the organization

Level 1 - No Tracking



- We know what TD is…
- …but we dont do much in our organization...

Level 2 – Ad-hoc tracking

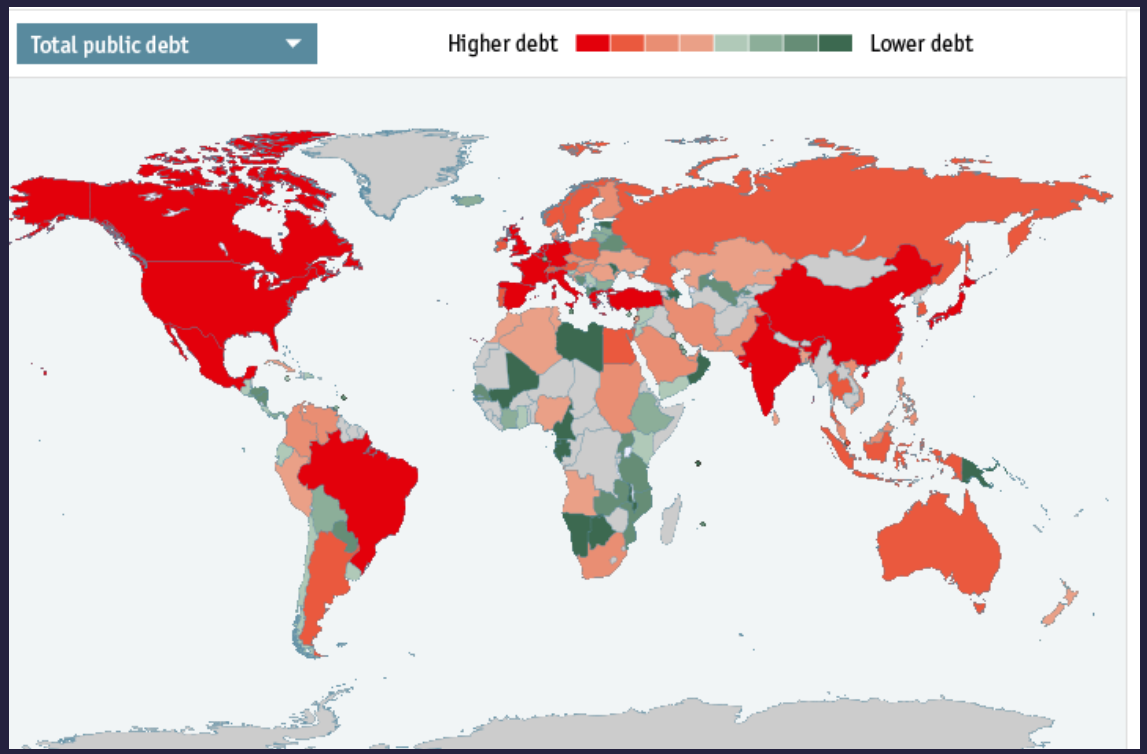


- No budget allocated
- Driven by individual initiative
- Use of “improper” tools

Level 3 – Systematic tracking

- Budget allocated to TD management
- TD-specific documentation
- Iterative process to monitor TD issues
- Continuous TD process improvement

Level 4 – Measured



- Use of automated TD identification tools
- Use of automated indicators to measure the interest of TD
- Integration of several measurements

Level 5 – Institutionalized

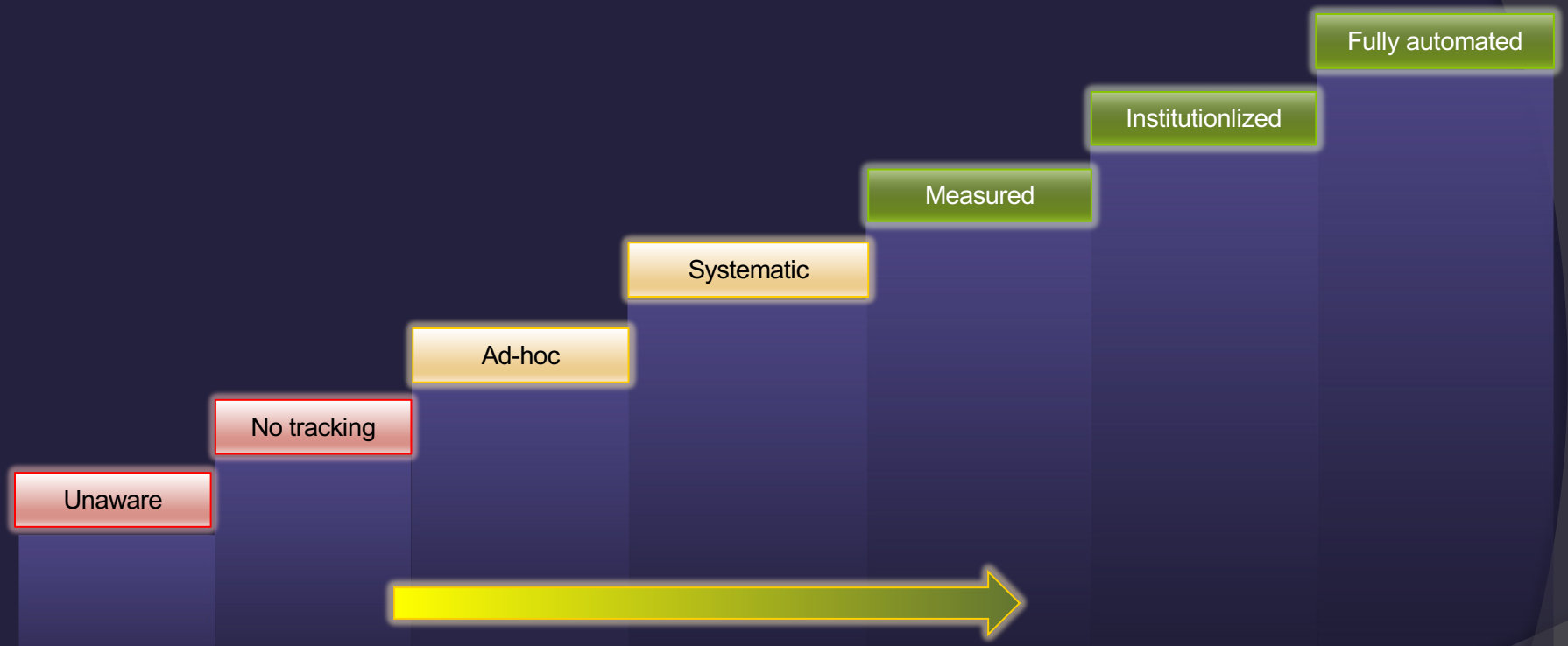
- Process standardized across the organization
- Spread to the whole organization
- Used for lifecycle planning
- Prioritization done among different kinds of TD

Level 6 – Fully automated

- Automated data-driven decisions
- statistical data from the history of the system
- assessment with existing benchmarks

- (Still a research work in progress)

Goal about Technical Debt Tracking*



* Martini, Antonio; Besker, Terese & Bosch, Jan (2018). Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming*.

Agile Technical Debt management

PROACTIVE APPROACHES

Avoid TD?

DAILY SCRUM MEETINGS

SPRINT CYCLE

CONTINUOUS • APPROACHES

PRODUCT BACKLOG

SPRINT BACKLOG

POTENTIALLY SHIPPABLE PRODUCT INCREMENT

Sprint X

Monitor TD?

Got TD?

Urgent?
Quick to fix?
High impact?

DAILY SCRUM MEETINGS

PRODUCT BACKLOG

REACTIVE APPROACHES

POTENTIALLY SHIPPABLE PRODUCT INCREMENT

Sprint X + 1

Proactive approaches

- Education
- Culture
- Organization
- Process
- Guidelines
- Visualization

Continuous approaches

- Semi-automatic Identification
- Code Reviews
- Retrospectives
- Technical leadership
- Dedicated refactoring sprints and % of time

Reactive approaches

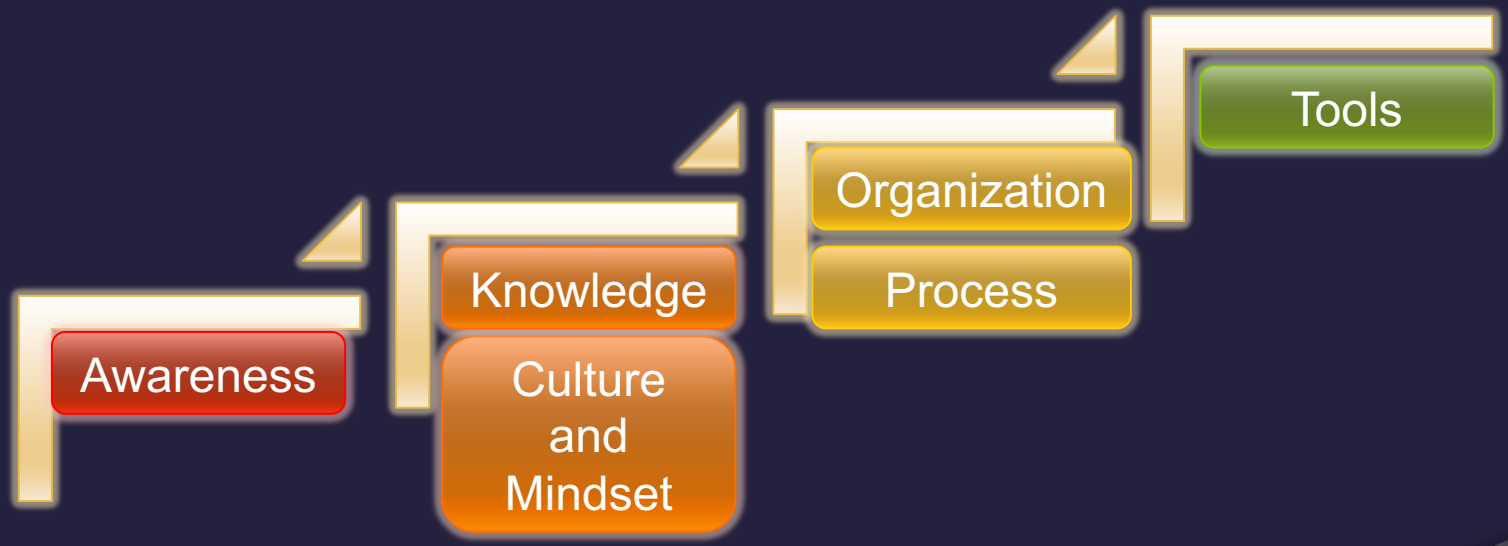
- Impact Map
- Roadmap evaluation
- Resources to remove TD
- Business case
- TD information used in planning and budget

- Agile: Refactoring

Proactive, continuous or reactive...?

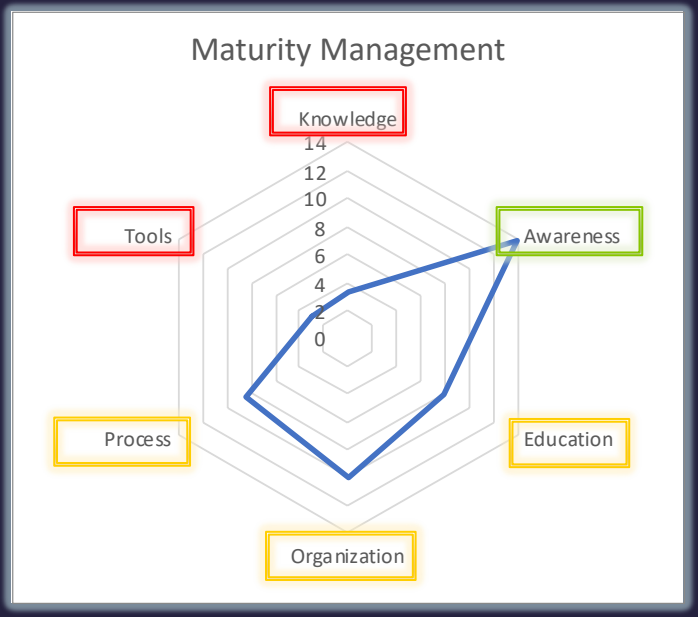
- We need all of them!
- Systematically managing TD means keeping an eye on different aspects

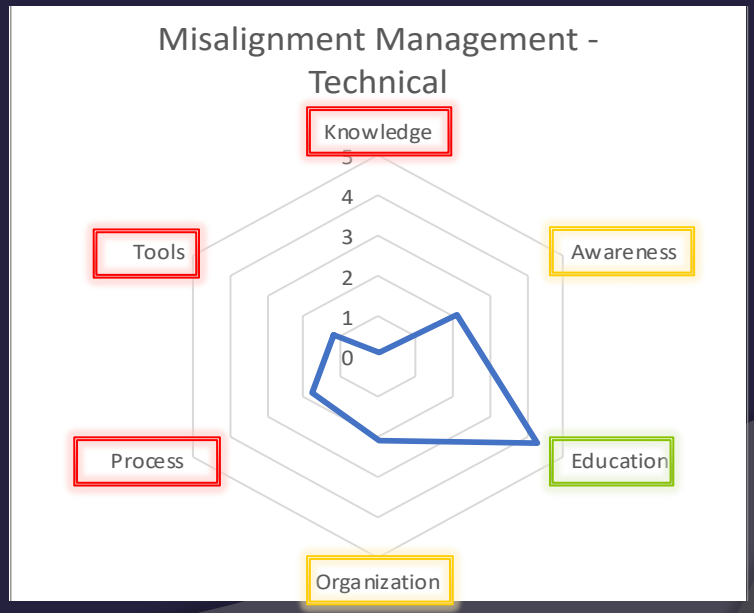
Climbing the TD management maturity ladder



Assessing TD management*

- Management maturity
- Alignment across different roles





* A. Martini, T. Besker, T. Posch and J. Bosch, "TD Pulse: Assessing the Systematic Management of Technical Debt," in IEEE Software, vol. 40, no. 3, pp. 54-62, May-June 2023