Architectural Technical Debt

- **Terese Besker**

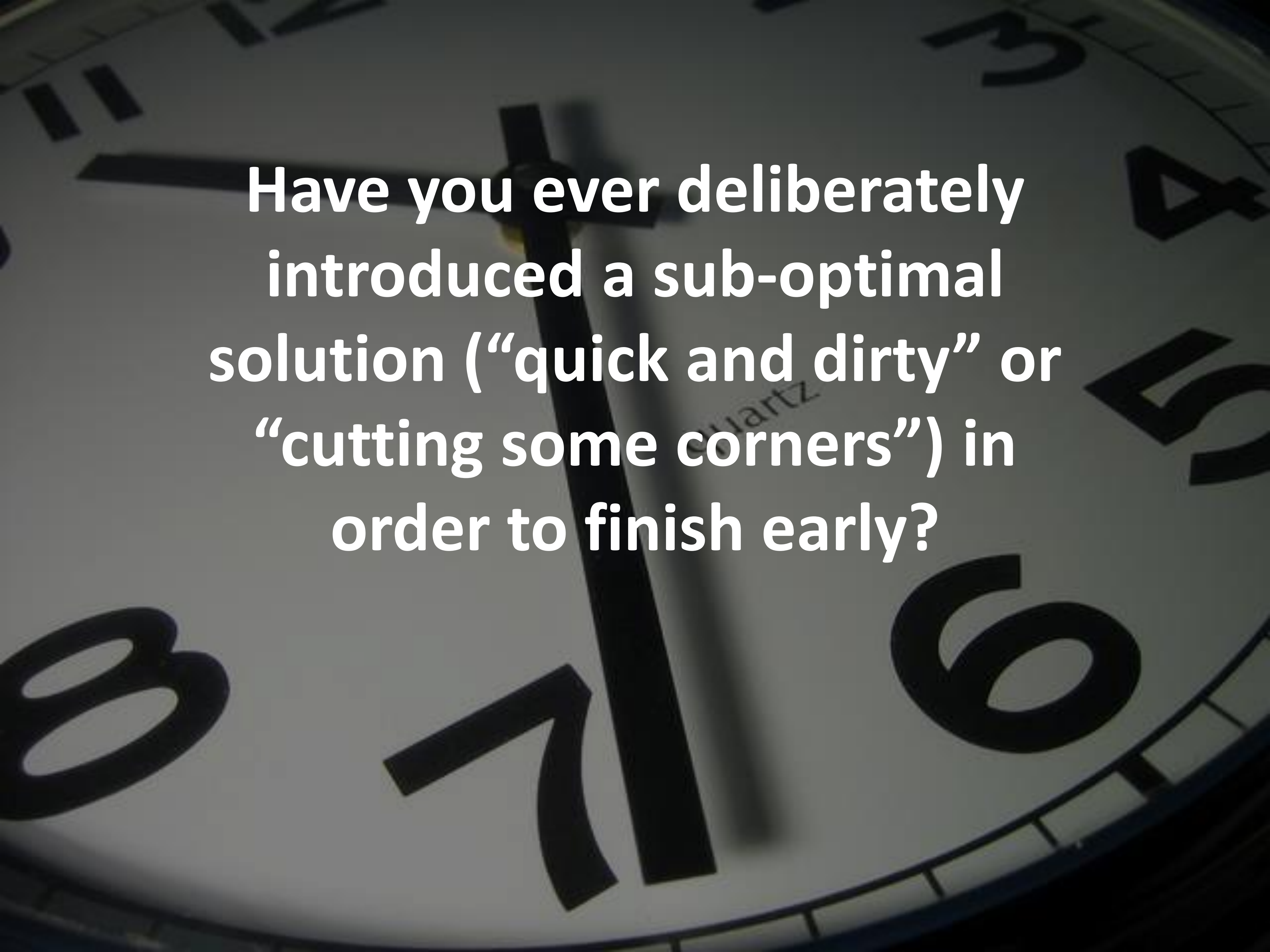
Software Center

- Chalmers University of Technology
Gothenburg, Sweden.
- www.software-center.se

Today’s excursion

- Cleaning in general
- Who am I?
- What is debt?
- What is Technical Debt?
- Solutions
- Prioritization
- Prevention mechanisms
- Regulations
- The costly Bill of Technical Debt
- Architectural Technical Debt
- Always a bad thing?
- A balancing act

Have you ever…..?



Have you ever deliberately introduced a sub-optimal solution (“quick and dirty” or “cutting some corners”) in order to finish early?

….or maybe you have introduced a sub-optimal solution *unconsciously* at some point?

Because you didn’t know any better way of doing it or…
it was the most optimal solution at that specific time

The final question:

Did you ever go back and fixed it?

Software Center

Mission: Improve the software engineering capability of the Nordic Software-Intensive industry with an order of magnitude

Theme: Fast, continuous deployment of customer value

Success: Academic excellence

Success: Industrial impact

Tetra Pak

SAAB TECHNOLOGIES

SIEMENS

verisure
ALARMS WITH IQ

AXIS®
COMMUNICATIONS

VOLVO

VOLVO

JEPPESEN®
A BOEING COMPANY

ERICSSON

GRUNDFOS

AVANCEZ 1829
CHALMERS

UNIVERSITAS GOTHOBURGENSIS 1891

MALMÖ UNIVERSITY

LINKÖPINGS UNIVERSITET

MÄLARDALEN UNIVERSITY
SWEDEN

A Debt Background

Financial debt:

- Loan
- Debt
- Interest

Basically a Debt is….

Borrowing against our capacity of tomorrow to make more progress today

What is *Technical* Debt?

A ordinary day at the software development office

Software Companies need to do:

Customer value

Software companies need to deliver customer value continuously, both from a *short- and long-term perspective*

Tradeoffs

Software companies need to consider the tradeoffs between the *overall quality* of the software, and the costs of the software development process in terms of the required *time and resources*

Efficiency

Software companies need to balance the quality of the software with the ambition of *increasing the efficiency* and *decreasing the costs* in each lifecycle phase

And what do we do?

Implement sub-optimal solutions *Deliberately*	*Deliberately* implement sub-optimal solutions in order to shorten the time-to-market or when resources are limited in practice, by implementing "quick fixes" or "cutting corners" during the software development process
Postponed refactoring tasks	Even if the best intention is to go back and refactor the sub-optimal solution immediately afterward, there is a tendency that these refactoring tasks will be postponed since, commonly, there are *other important deadlines in the near future*, where these refactoring tasks are often down-prioritized
Implement sub-optimal solutions *Unintentionally*	There is also the scenario where sub-optimal solutions are implemented *unintentionally*, due to a lack of knowledge, guidelines or best practices.

Eavesdropping at the office

Let's finish the testing in the next release*

Let's just copy and paste this part*

We don't have time to reconcile these two databases right now, lets use some glue code and we can fix it later

Lets do a quick and dirty solution now, and we can have a look at this in next sprint(s)

* R. K. Gupta, P. Manikreddy, S. Naik, and K. Arya, "Pragmatic Approach for Managing Technical Debt in Legacy Software Project," in Proceedings of the 9th India Software Engineering Conference, Goa, India, 2016, pp. 170-176.

"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…”

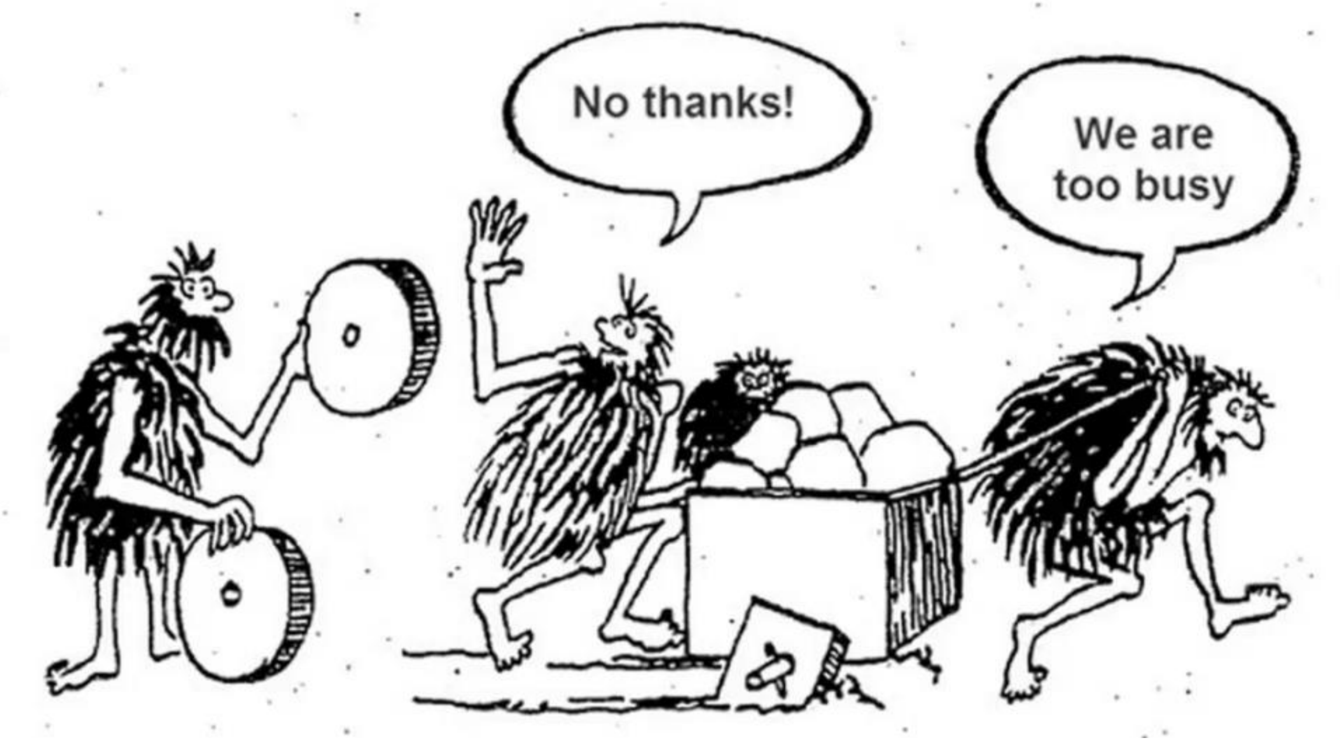
“Every minute spent on not-quite-right code counts as interest on that debt”

Ward Cunningham

Why the Technical Debt metaphor?

Helps business staff to understand and make technical decisions

Helps technical staff to understand financial consequences of technical decisions, and argue for e.g. the need for refactoring



Technical Debt is **sub-optimal solutions**, **not bugs**, and **not yet implemented features**

Technical Debt

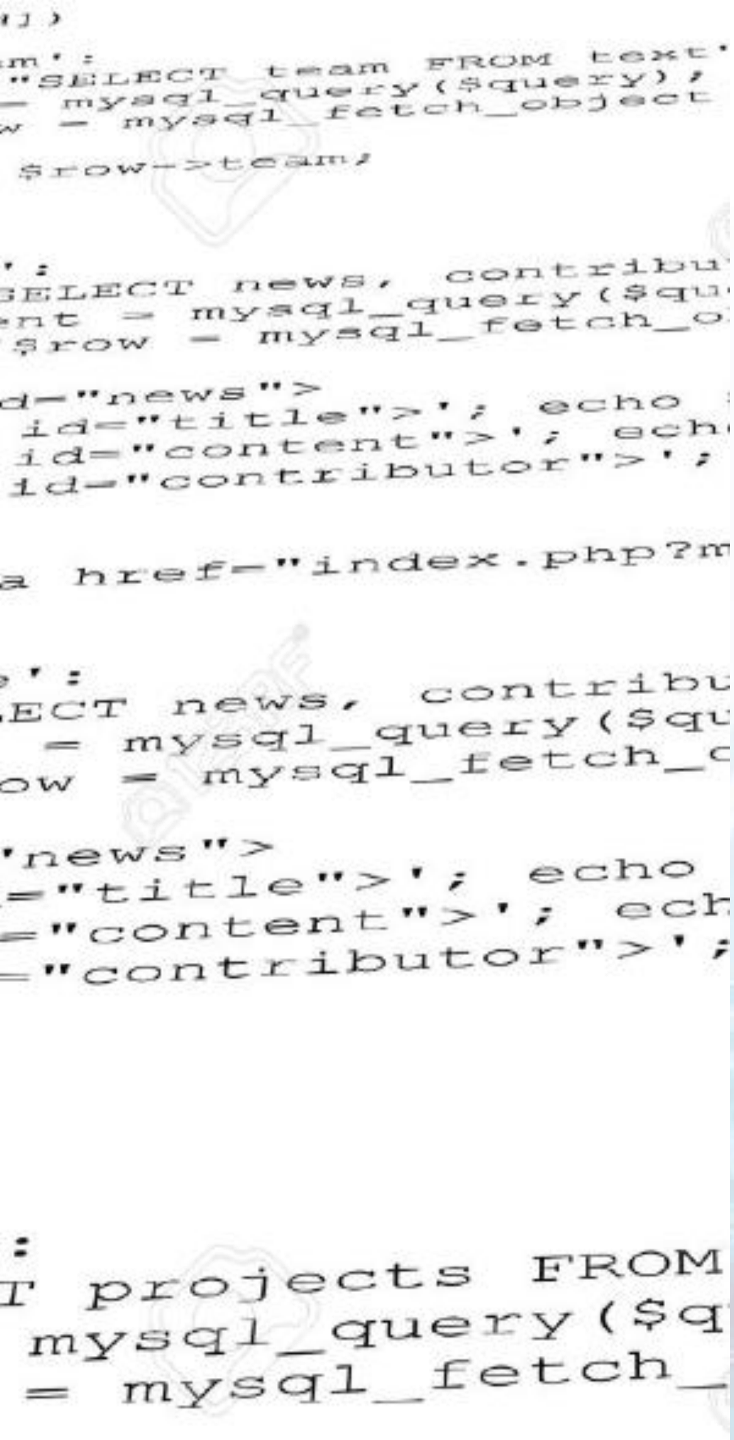
Customer's view

Developer's view

Technical Debt, Features, Defects, etc.

	Visible	Invisible
Positive Value	New Features Added Functionality	Architectural, Structural Features
Negative Value	Defects	Technical Debt

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



```
'])
m':
"SELECT team FROM text'
= mysql_query($query);
w = mysql_fetch_object

$row->team;

':
SELECT news, contribu
nt = mysql_query($qu
$row = mysql_fetch_o

d="news">
id="title">'; echo
id="content">'; ech
id="contributor">';

a href="index.php?m

':
ECT news, contribu
= mysql_query($qu
ow = mysql_fetch_c

'news">
="title">'; echo
="content">'; ech
="contributor">';

:
T projects FROM
mysql_query($q
= mysql_fetch_
```

Examples of Technical Debt

- **Poor code quality**
- **Poor or inappropriate Architecture of the software**
- **Lack of following guidelines**
- **Lack of documentation**
- **Lack of testing**
- **… and so on**

The Bill: Software related Interest

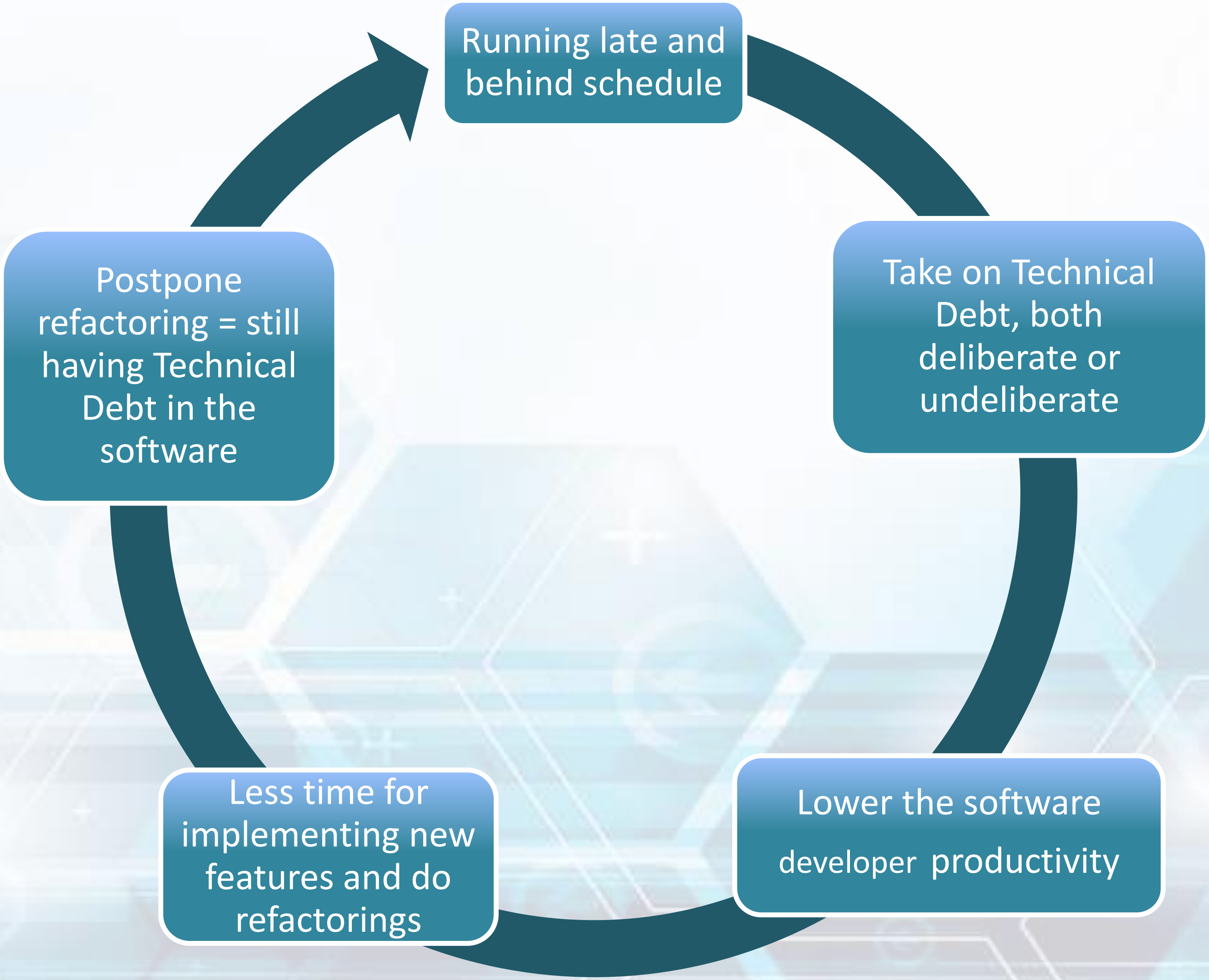
- it is not technical debt if you don't pay any interest

- Working around or fixing existing errors
- Extra effort spent on understanding complex code
- Baby-sitting tasks that could be automated

Worst case scenarios

- Impede innovation and expansion of your software systems
- Stifle an whole organization’s ability to innovate
- Negative effect on available resources to implement new technology
- Time consuming maintenance
- Lower the productivity
- Lower the morale
- In the long run, it can **lead to a system crisis**

Examples of Vicious circles



Running late and behind schedule

Take on Technical Debt, both deliberate or undeliberate

Lower the software developer productivity

Less time for implementing new features and do refactorings

Postpone refactoring = still having Technical Debt in the software

Suddenly, you reach a point when you have to take a step back and reflect on where you are going

Solutions…

- **Refactoring** – to optimize (and clean) the software without impacting the user experience or functionality

- **Continuous process** of refactoring initiatives

- Refactoring is crucial to prevent spiraling technical debt

Remediation of Technical Debt

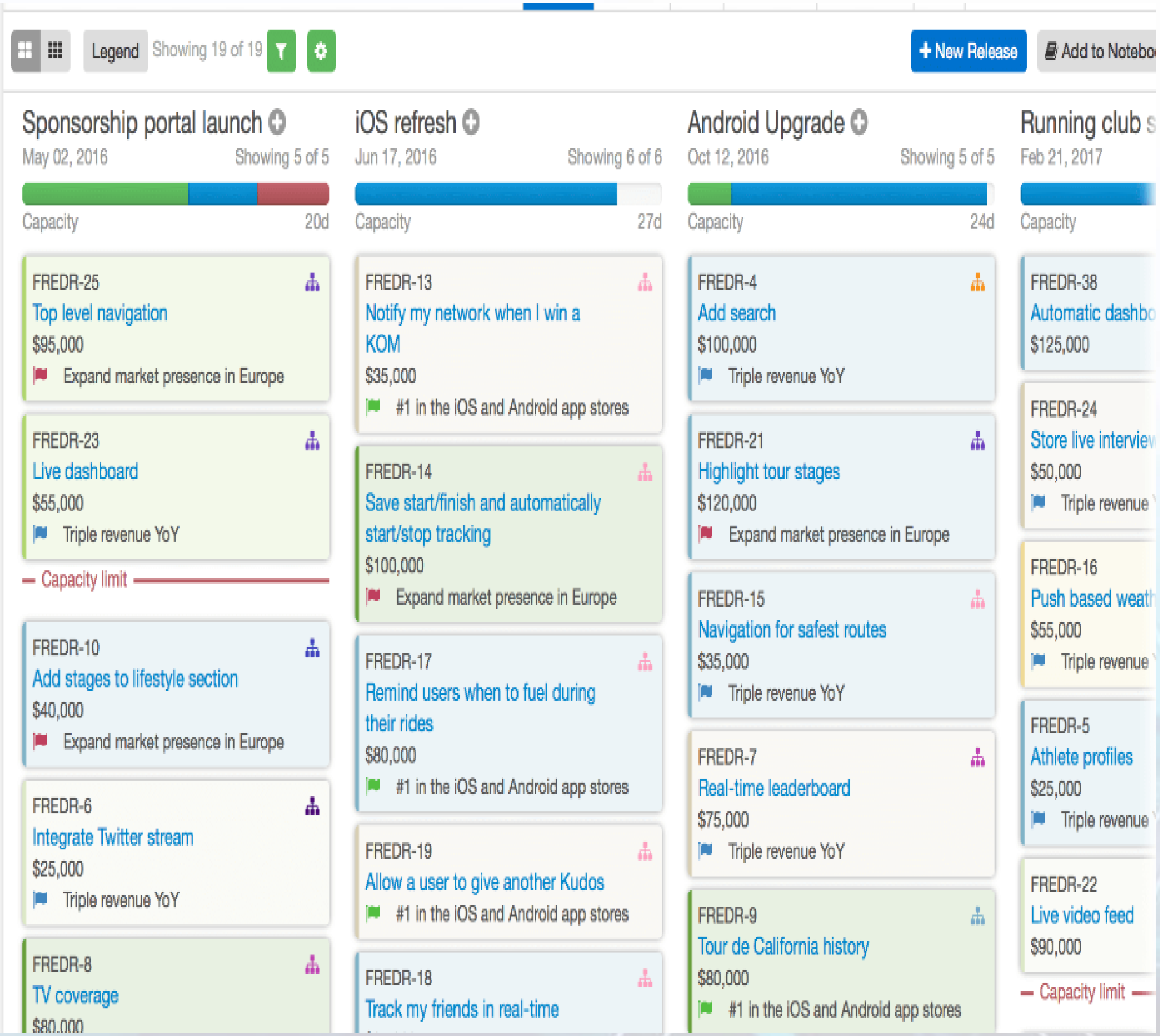
- **The only significantly effective way of reducing TD, is to *refactor* it**

- **Refactoring activities of the identified TD items needs to be *prioritized***

- ***Competing* with for example implementation of new features**

Besker, T., et al. (2019). Technical debt triage in backlog management. Proceedings of the Second International Conference on Technical Debt. Montreal, Quebec, Canada, IEEE Press**:** 13-22.

The Agile Backlog



Sponsorship portal launch
May 02, 2016 Showing 5 of 5
Capacity 20d

FREDR-25
Top level navigation
$95,000
Expand market presence in Europe

FREDR-23
Live dashboard
$55,000
Triple revenue YoY

Capacity limit

FREDR-10
Add stages to lifestyle section
$40,000
Expand market presence in Europe

FREDR-6
Integrate Twitter stream
$25,000
Triple revenue YoY

FREDR-8
TV coverage
$80,000

iOS refresh
Jun 17, 2016 Showing 6 of 6
Capacity 27d

FREDR-13
Notify my network when I win a KOM
$35,000
#1 in the iOS and Android app stores

FREDR-14
Save start/finish and automatically start/stop tracking
$100,000
Expand market presence in Europe

FREDR-17
Remind users when to fuel during their rides
$80,000
#1 in the iOS and Android app stores

FREDR-19
Allow a user to give another Kudos
#1 in the iOS and Android app stores

FREDR-18
Track my friends in real-time

Android Upgrade
Oct 12, 2016 Showing 5 of 5
Capacity 24d

FREDR-4
Add search
$100,000
Triple revenue YoY

FREDR-21
Highlight tour stages
$120,000
Expand market presence in Europe

FREDR-15
Navigation for safest routes
$35,000
Triple revenue YoY

FREDR-7
Real-time leaderboard
$75,000
Triple revenue YoY

FREDR-9
Tour de California history
$80,000
#1 in the iOS and Android app stores

Running club s
Feb 21, 2017
Capacity

FREDR-38
Automatic dashbo
$125,000

FREDR-24
Store live intervie
$50,000
Triple revenue

FREDR-16
Push based weath
$55,000
Triple revenue

FREDR-5
Athlete profiles
$25,000
Triple revenue

FREDR-22
Live video feed
$90,000

Capacity limit

The presence of Technical Debt items in backlogs?

Does the Prioritization process of the backlog also include the prioritization of Technical Debt?

Technical Debt issues – not in same Backlog as Features and Bug fixes

Technical Debt issues – in a “***shadow*** “ backlog

Fixed amount of time in each sprint allocated for improvement, which includes TD, however no follow-up on time spent

Besker, T., et al. (2019). Technical debt triage in backlog management. Proceedings of the Second International Conference on Technical Debt. Montreal, Quebec, Canada, IEEE Press**:** 13-22.

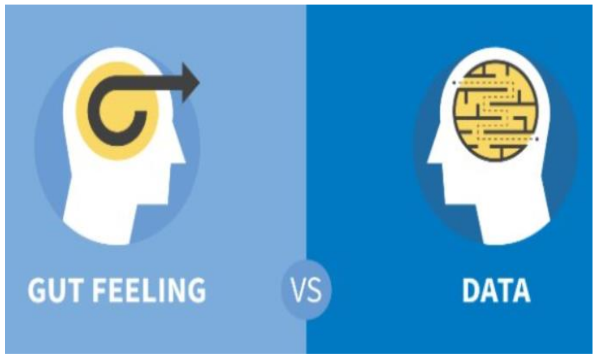
Supporting frameworks or Gut Feelings?

Company commonly does not use any guiding Decision Making Frameworks

"In my experience, it's usually the most experienced guy that has the biggest impact [when prioritizing TD]. We don't actually need a big consensus among the participants."

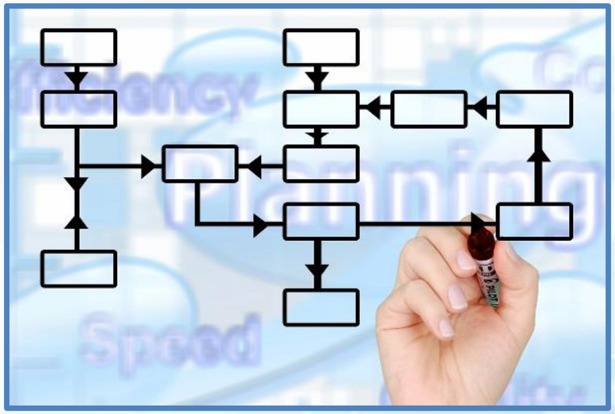
Gut feeling is not an add-hoc approach:

- **Prior experience**
- **Acquired knowledge**
- **Instinct or emotion**
- **Roadmap of future features**
- **……**



A reactive or proactive approach

The prioritization of TD in the backlog is much more of a **reactive** then a proactive approach

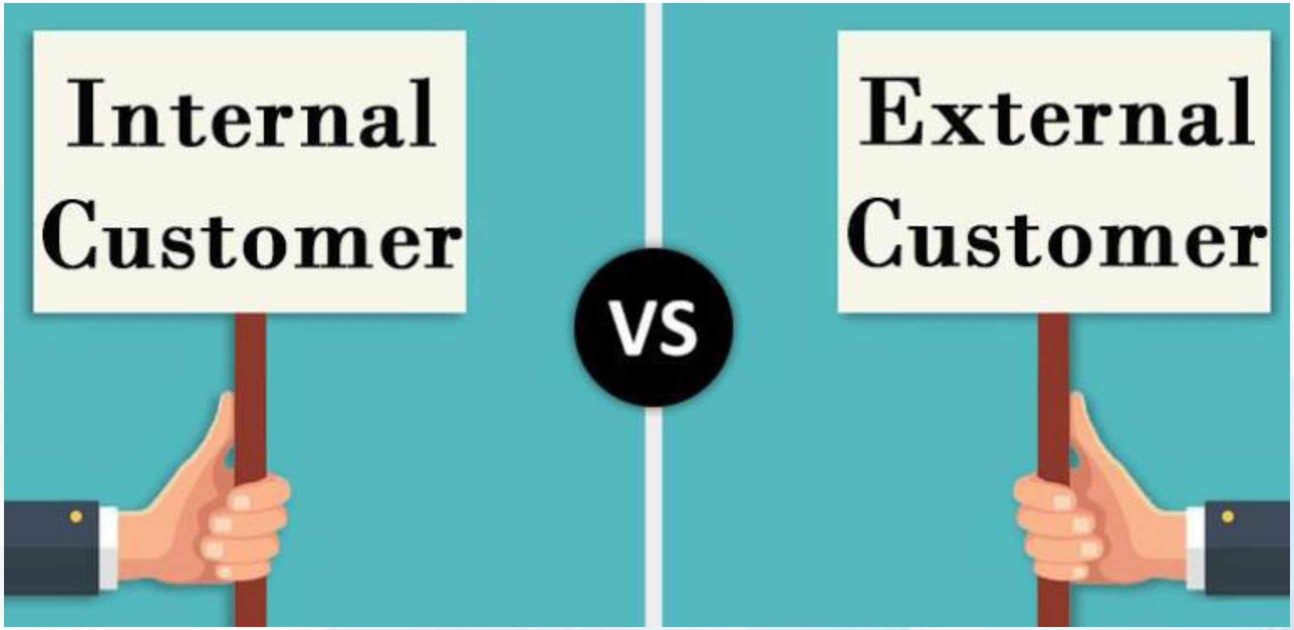


Estimating the value of doing refactoring of Technical Debt, is considered to be difficult

92 % states that the Technical Debt's negative effects could be reduced, if they did the prioritization of their Backlog differently

Besker, T., et al. (2019). Technical debt triage in backlog management. Proceedings of the Second International Conference on Technical Debt. Montreal, Quebec, Canada, IEEE Press**:** 13-22.

TD refactoring competition with customer requirements



92 % states that the Technical Debt's negative effects could be reduced, if they did the prioritization of their Backlog differently

Besker, T., et al. (2019). Technical debt triage in backlog management. Proceedings of the Second International Conference on Technical Debt. Montreal, Quebec, Canada, IEEE Press**:** 13-22.

Refactoring contra Prevention

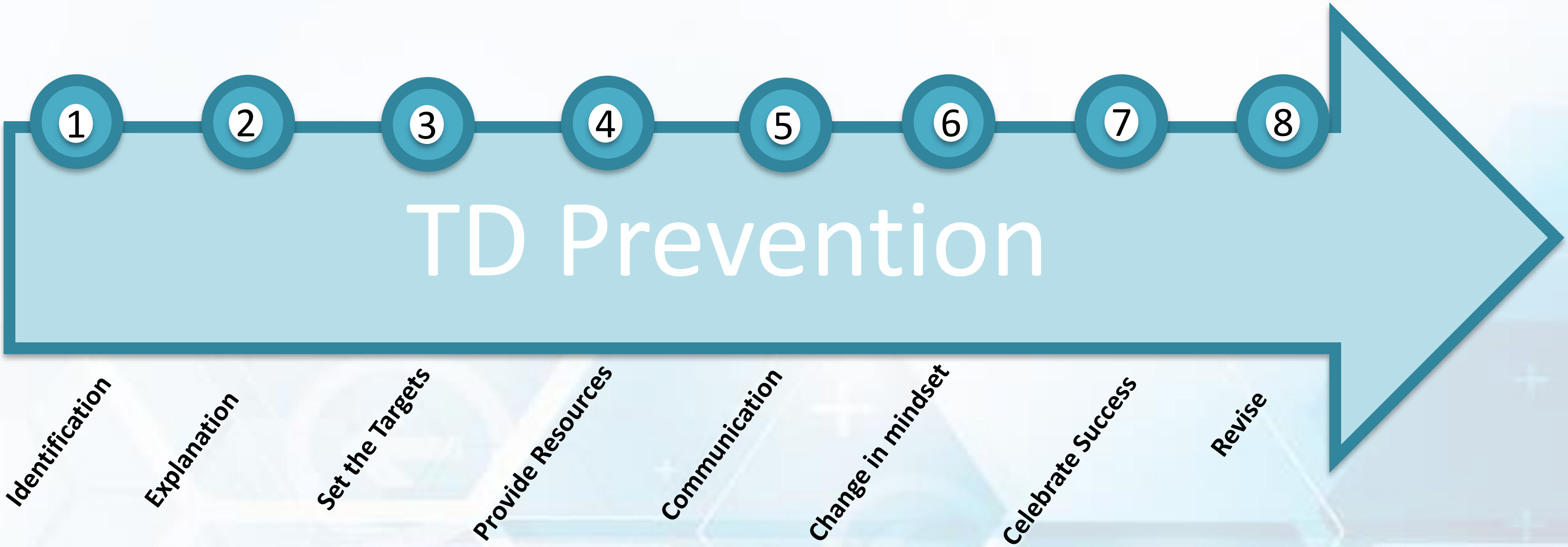
Besides continuously refactoring activities we also need to prevent introducing Technical Debt from the very beginning

TD prevention……prevents potential TD from being incurred, in the first place

- Commonly TD Prevention is "cheaper" than TD repayment
- There is no tool for TD prevention - > **Development Process Improvement**

TD prevention –

from a Change Management perspective



TD prevention (1/2)

Identify what need to be improved

coding standards

code reviews

definition of done

architectural structure (e.g. Monolithic or Micro Services)

Explain the cost and nature of debt to developers architect, PO, PM etc.

Debt awareness is best among the methods of debt prevention

Harmfulness today and in future (predicting growth of interest costs)

Productivity increase

Feel more confident (developers pride) and attract the "best" developers

Set the Targets (clear steps with measurable targets e.g. wasted time)

Provide Resources (tools such as AnaConDebt, SonarQube, Arcan, education etc.)

TD prevention (2/2)

Communication

Change in mindset

Manage resistance and cultivate a culture

Celebrate Success

Recognizing milestone achievements

Encouragement

Review, Revise and Continuously Improve

Can regulations stop us from introducing TD in the first place?

The relationship between safety-critical software (SCS) regulations and the management of TD

Examples of regulatory certification processes

SCS are heavily regulated

SCS require certification against industry standards

Recertified to ensure compliance with the present safety standards.

E.g. after a refactoring activity of software:

retested

revalidated

reverified

Cost and time-consuming – risk of being down-prioritized or avoided -> more TD

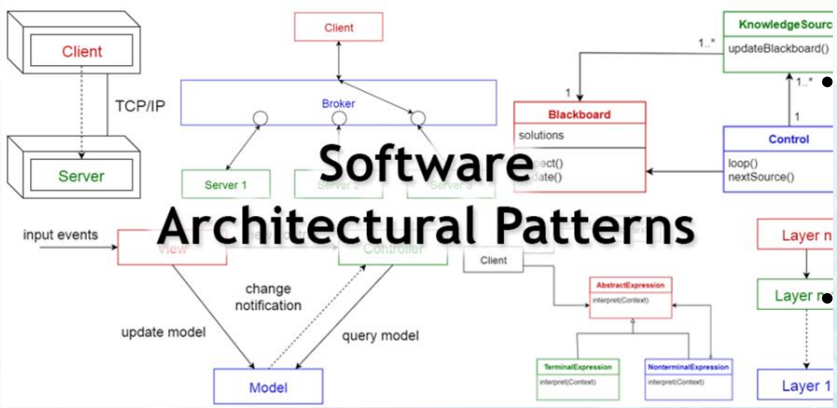
Besker, T., et al. (2018). How Regulations of Safety-Critical Software Affect Technical Debt. 2019, SEAA

Consequences and Effects of SCS Regulations when Conducting or Planning for TD Refactoring Activities

TD refactoring activities are commonly deliberately avoided

Besker, T., et al. (2018). How Regulations of Safety-Critical Software Affect Technical Debt. 2019, SEAA

Software Architectural Structures Contributing to TD Refactoring



- **Components can have different levels of safety regulations, which defines the refactoring scope**

- **The importance of a software architecture that facilitates refactoring with as little effort and cost as possible**

- **Examples of different architectural structures; component-based, pipes and filters, monolithic, and layered structures**

Besker, T., et al. (2018). How Regulations of Safety-Critical Software Affect Technical Debt. 2019, SEAA

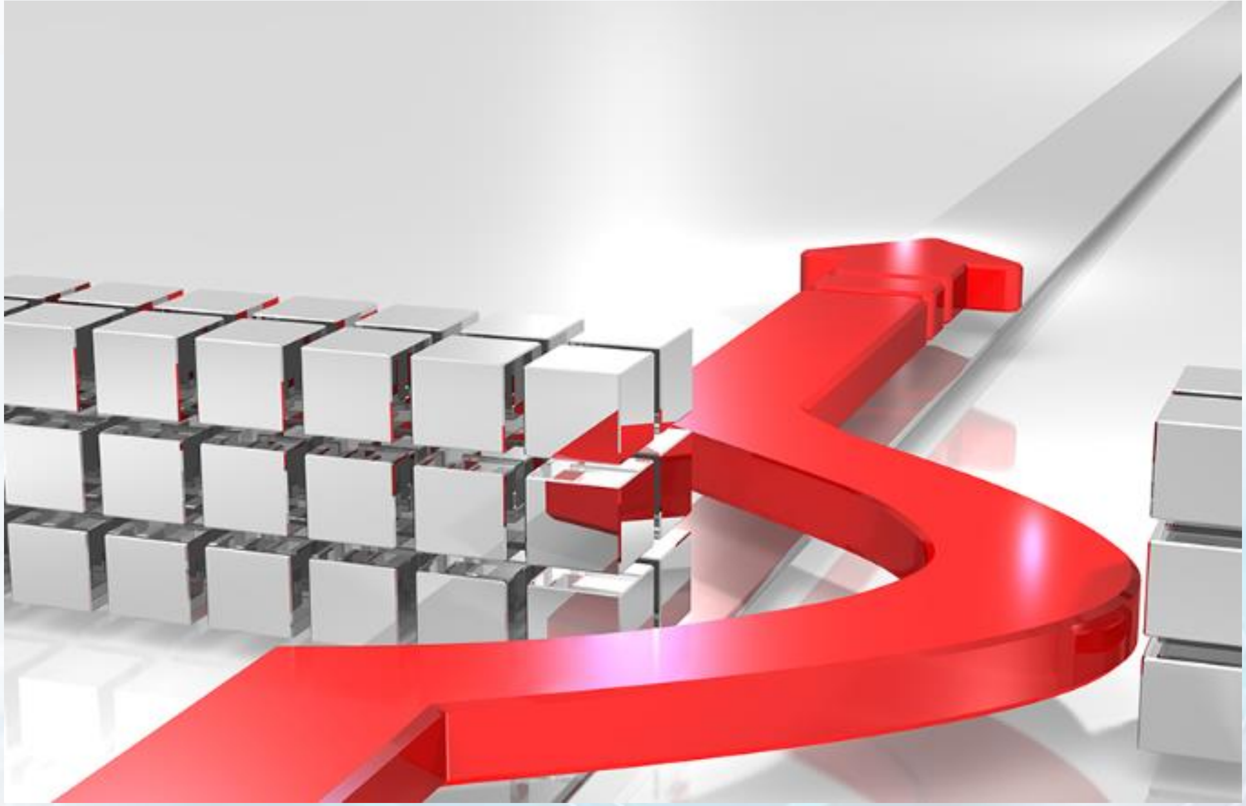
Software Architectural Structures Contributing to TD Refactoring

- **Monolithic architecture = major hindrance for TD refactoring tasks in SCS**

- **Modular SCS architecture** (component-based or loosely coupled units or layer-based structures) **= increase likelihood of TD Refactoring tasks**

"Our middle layer in the architecture would have looked different [if it was not SCS] since the intention of the decision level is actually to abstract and isolate different ASIL levels because it would be quite hard and expensive to maintain these dependencies otherwise."

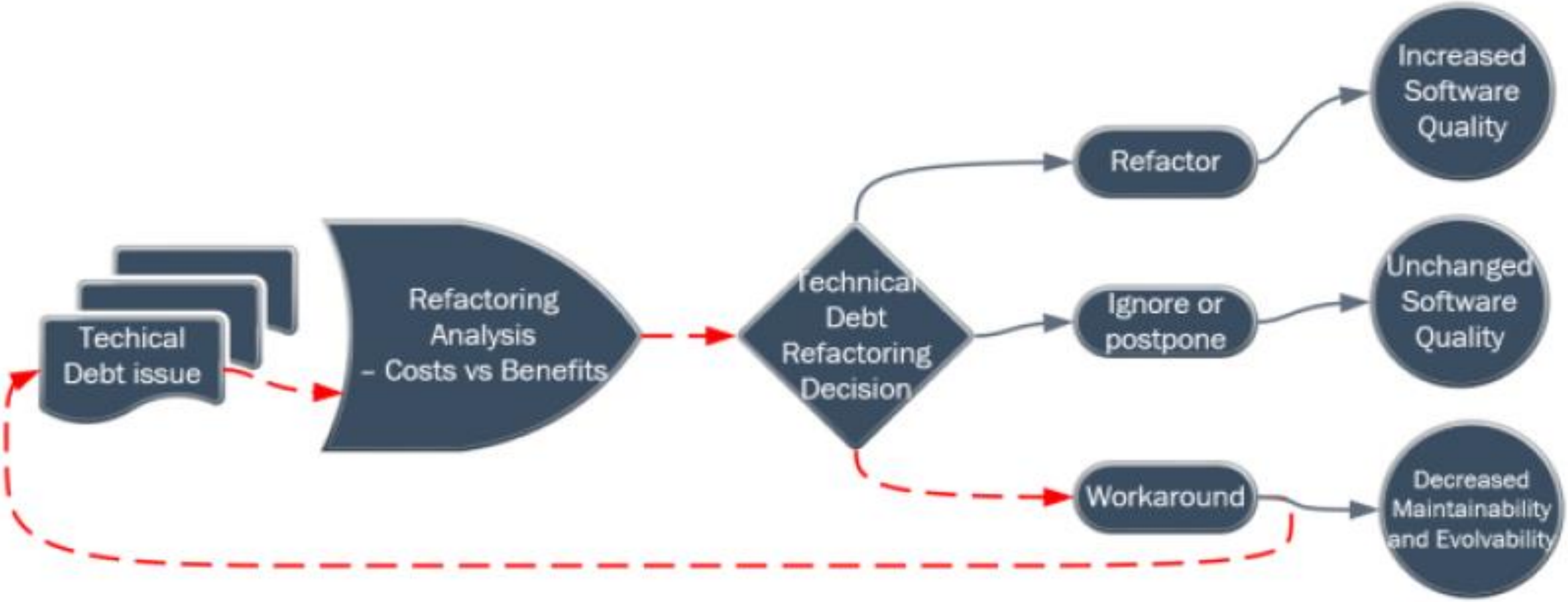
Consequences and Effects of SCS Regulations when Conducting or Planning for TD Refactoring Activities



Work-around solutions to avoid the additional activities and costs

Besker, T., et al. (2018). How Regulations of Safety-Critical Software Affect Technical Debt. 2019, SEAA

The Counterproductiveness of the SCS Regulations


Techical Debt issue
Refactoring Analysis - Costs vs Benefits
Technical Debt Refactoring Decision
Refactor
Ignore or postpone
Workaround
Increased Software Quality
Unchanged Software Quality
Decreased Maintainability and Evolvability

Even if the SCS regulations have the best intention to produce a high-quality software product, the findings demonstrate that these heavy regulations are conceivably counterproductive since they potentially can constrain the possibility of performing optimal TD refactoring activities efficiently

Opposite effect : the regulations contribute to the further introduction of TD and thereby potentially decrease both the maintainability and evolvability of the software.

Besker, T., et al. (2018). How Regulations of Safety-Critical Software Affect Technical Debt. 2019, SEAA

How expensive is Technical Debt? – from a productivity perspective

- Technical Debt cause developers to **waste working time**, since they have to perform extra activities due to the present Technical Debt.

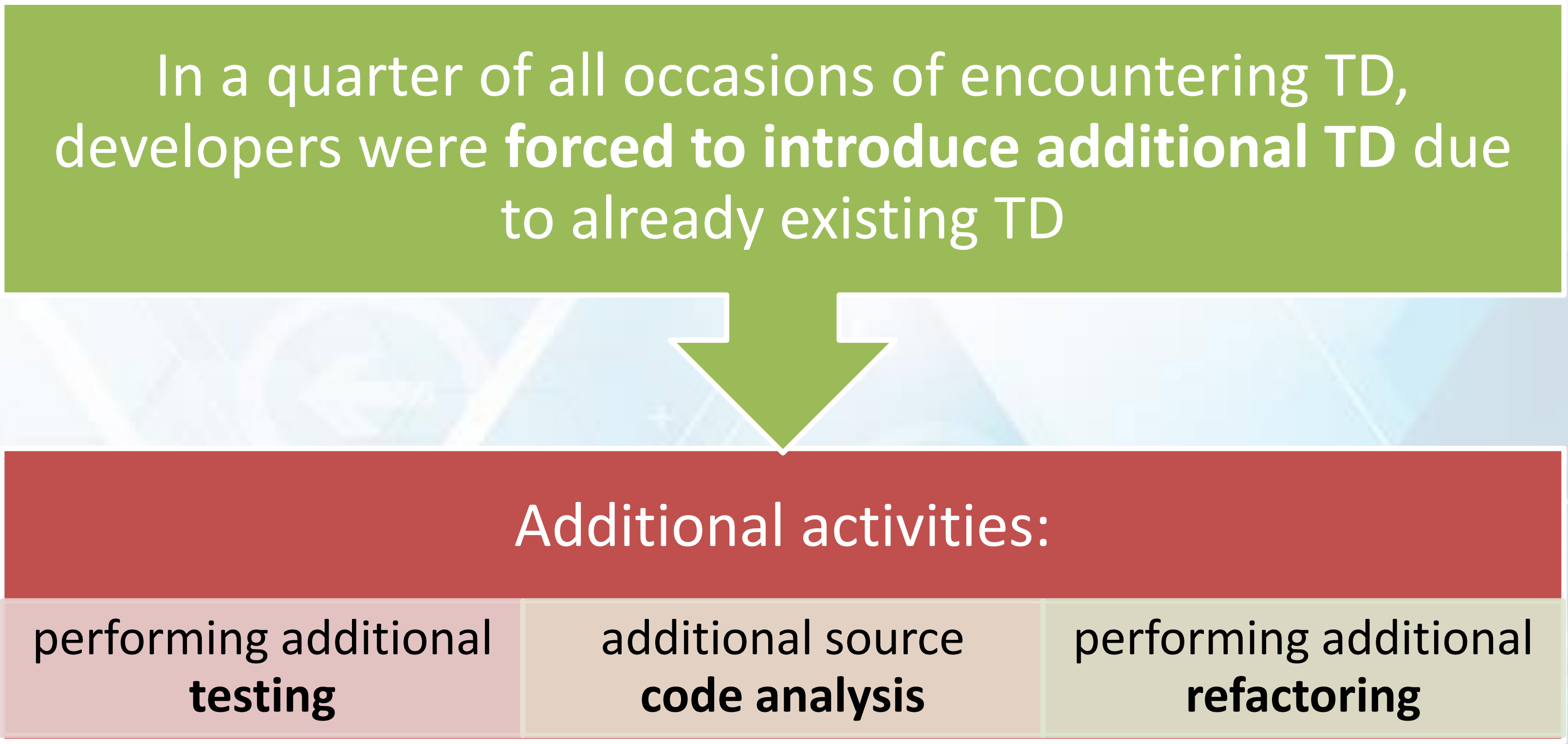
- **How much time?**

Technical Debt contra Productivity

24 % of all development time is wasted by developers, due to Technical Debt

Besker, T., et al. (2019). "Software developer productivity loss due to technical debt—A replication and extension study examining developers’ development work." Journal of Systems and Software **156**: 41-61.

Technical Debt contra Productivity



In a quarter of all occasions of encountering TD, developers were **forced to introduce additional TD** due to already existing TD

Additional activities:

performing additional **testing**

additional source **code analysis**

performing additional **refactoring**

Besker, T., et al. (2019). "Software developer productivity loss due to technical debt—A replication and extension study examining developers' development work." Journal of Systems and Software **156**: 41-61.

Technical Debt and Morale

- TD can reduce developers' morale; the presence of TD hinders developers from performing their tasks and achieving their goals

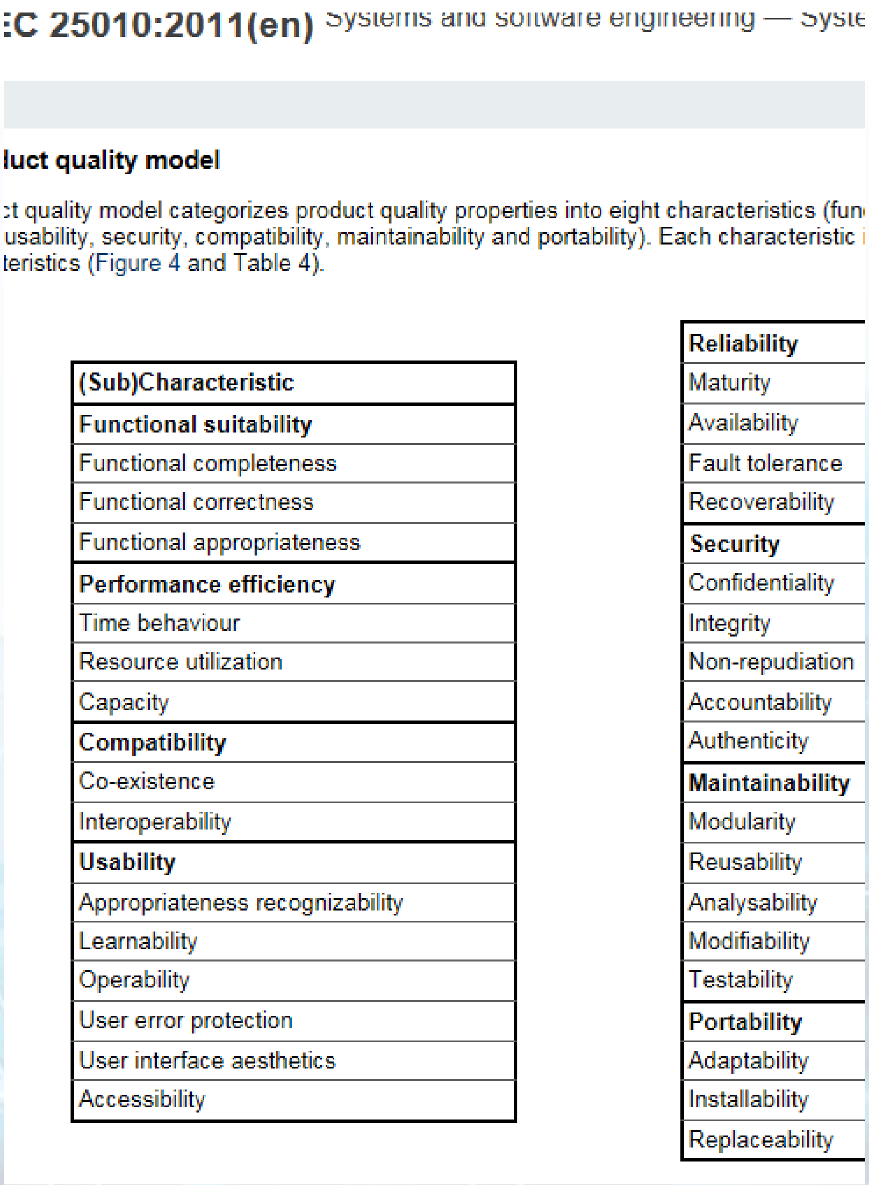
- A proper management of TD increases developers' morale

Ghanbari, H., et al. (2017). Looking for Peace of Mind? Manage your (Technical) Debt - An Exploratory Field Study. 11th International Symposium On Empirical Engineering and Measurement (ESEM), Toronto, Canada.

What about the Software Quality due to having Technical Debt?

Compromised quality attributes due to Technical Debt

- *Maintainability*
- *Reliability*
- *Performance*
- *Reusability*
- *Ability to add new features*



C 25010:2011(en) Systems and software engineering — Syste

luct quality model

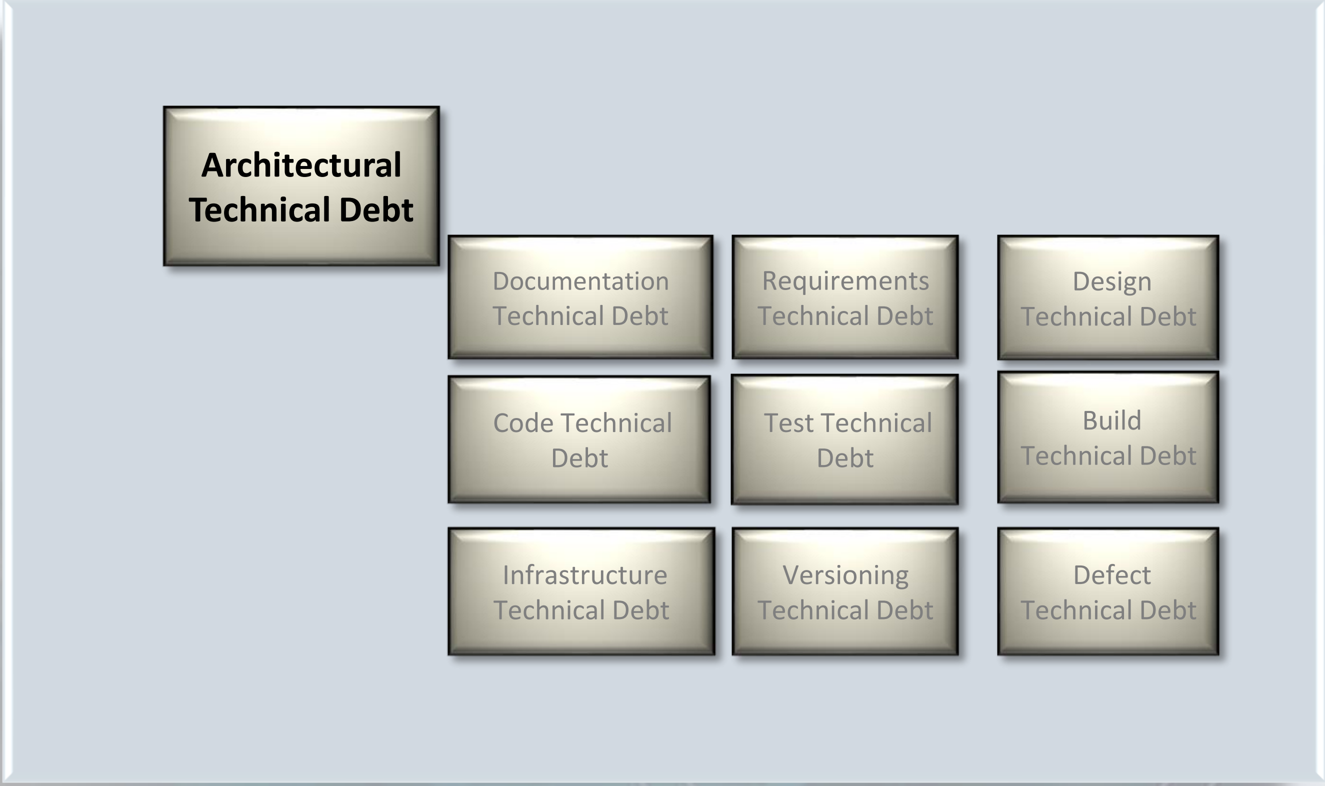
t quality model categorizes product quality properties into eight characteristics (fun usability, security, compatibility, maintainability and portability). Each characteristic i teristics (Figure 4 and Table 4).

(Sub)Characteristic
Functional suitability
Functional completeness
Functional correctness
Functional appropriateness
Performance efficiency
Time behaviour
Resource utilization
Capacity
Compatibility
Co-existence
Interoperability
Usability
Appropriateness recognizability
Learnability
Operability
User error protection
User interface aesthetics
Accessibility

Reliability
Maturity
Availability
Fault tolerance
Recoverability
Security
Confidentiality
Integrity
Non-repudiation
Accountability
Authenticity
Maintainability
Modularity
Reusability
Analysability
Modifiability
Testability
Portability
Adaptability
Installability
Replaceability

Besker, T., et al. (2017). Time to Pay Up - Technical Debt from a Software Quality Perspective. proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ ICSE17, Buenos Aires, Argentina, CibSE.

Different types of Technical Debt*



Architectural Technical Debt

Documentation Technical Debt

Requirements Technical Debt

Design Technical Debt

Code Technical Debt

Test Technical Debt

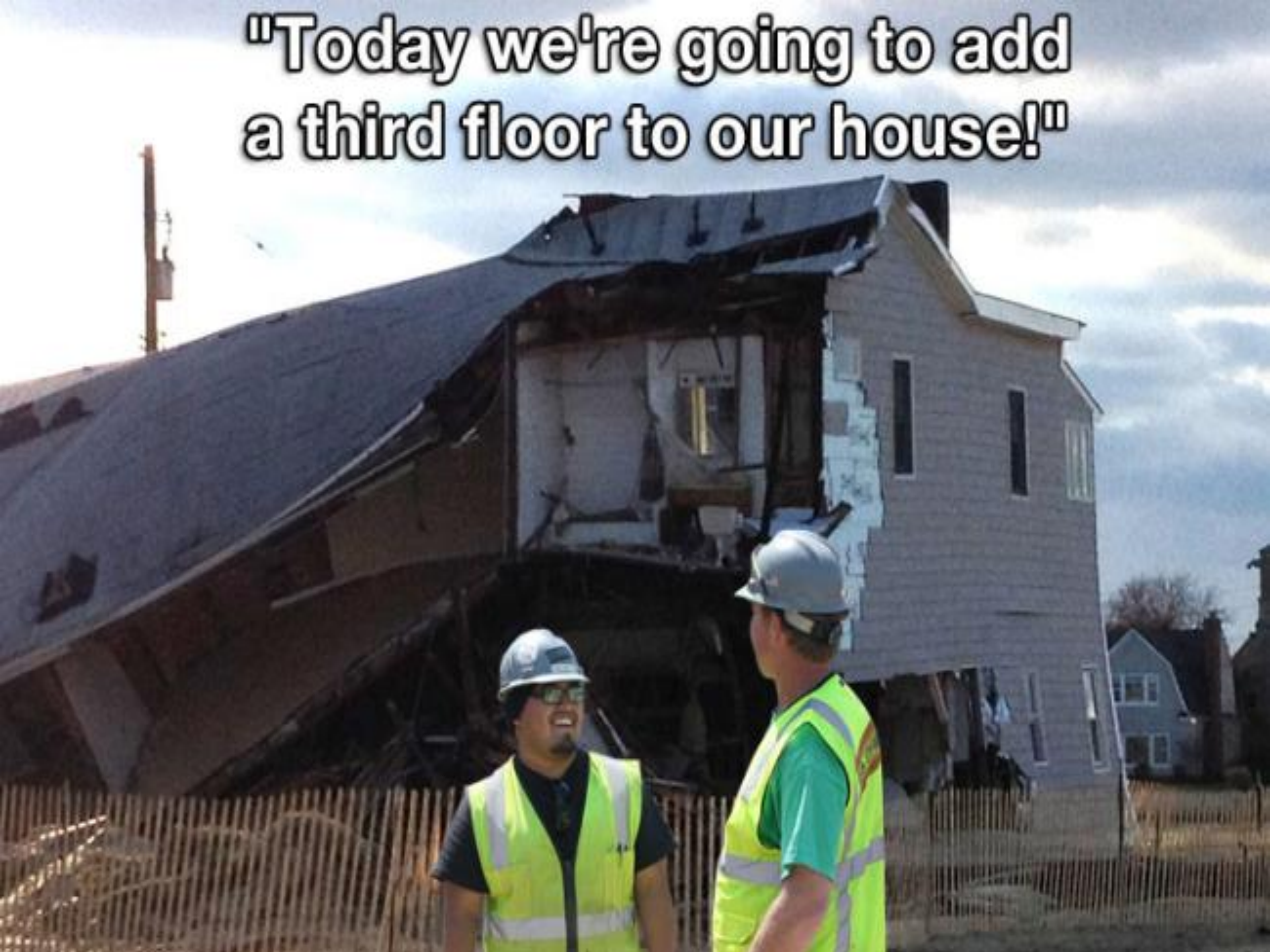
Build Technical Debt

Infrastructure Technical Debt

Versioning Technical Debt

Defect Technical Debt

* E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” Journal of Systems and Software, vol. 86, no. 6, 2013, pp. 1498-1516.



"Today we're going to add a third floor to our house!"

Architectural Technical Debt – ATD

The importance of ATD

- ATD is the **most commonly encountered** instances of TD and are caused by architectural inadequacies

- Architectural decisions are the **most important source of TD**

- ATD has a **huge impact and leverage** within the overall development lifecycle

- *“Architecture plays a significant role in the development of large systems, together with other development activities, such as documentation and testing (which are often lacking). These activities can add significantly to the debt and thus are part of the technical debt landscape”. ***

** Kruchten, P. , Nord, R.L. , Ozkaya, I. , 2012. Technical debt: from metaphor to theory and practice. Software, IEEE 29, 18–21 .

Categories related to ATD

Dependencies violations, including module dependencies, external dependencies, and external team dependencies

Non-uniformity of patterns and policies where, for example, a violation of naming conventions and non-uniform design or architectural patterns are implemented

Code-related issues such as code **duplication** and overly **complex code**

Non-uniform management of **integration with subsystems** and resources

Conflicting QA synergies

Besker, T., et al. (2018). "Managing architectural technical debt: A unified model and systematic literature review." Journal of Systems and Software **135**(Supplement C): 1-16.

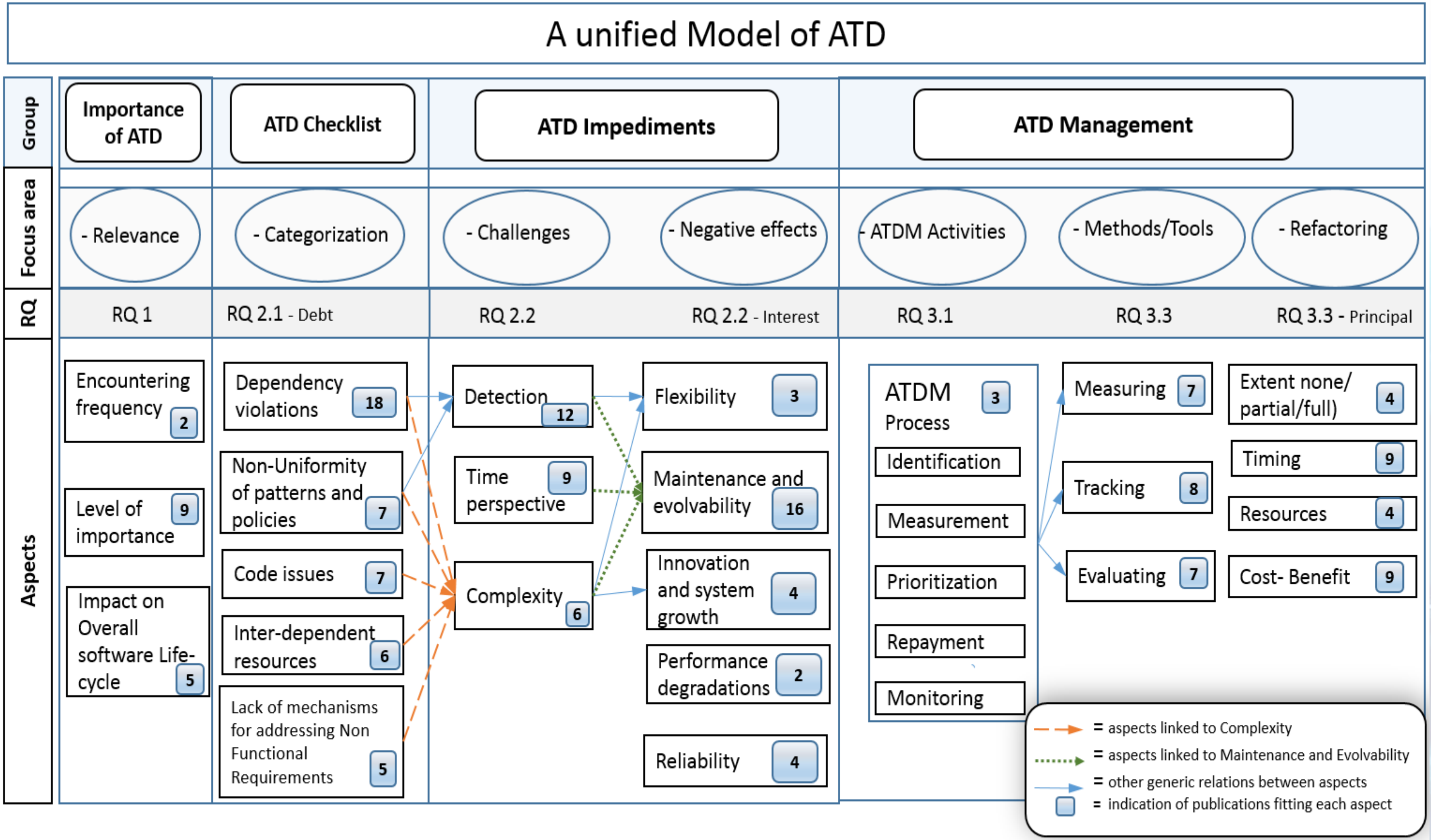
Challenges related to ATD

- **Detection, no available tools supporting the detection of ATD**
- **ATD seldom yield observable behaviors to end users**
- **ATD evolves over time**

Negative effects caused by ATD

- **Reduced flexibility – need for a proactive thinking**
- **Maintenance complications and penalties**
- **Stifling the organization's ability to introduce new features**
- **Imped innovation and system growth (evolvability, extendability)**
- **Understandability, testability, extensibility, reusability performance and reliability**

Architectural Technical Debt



A unified Model of ATD

Group | Focus area | RQ | Aspects

Importance of ATD
- Relevance
RQ 1
Encountering frequency 2
Level of importance 9
Impact on Overall software Life-cycle 5

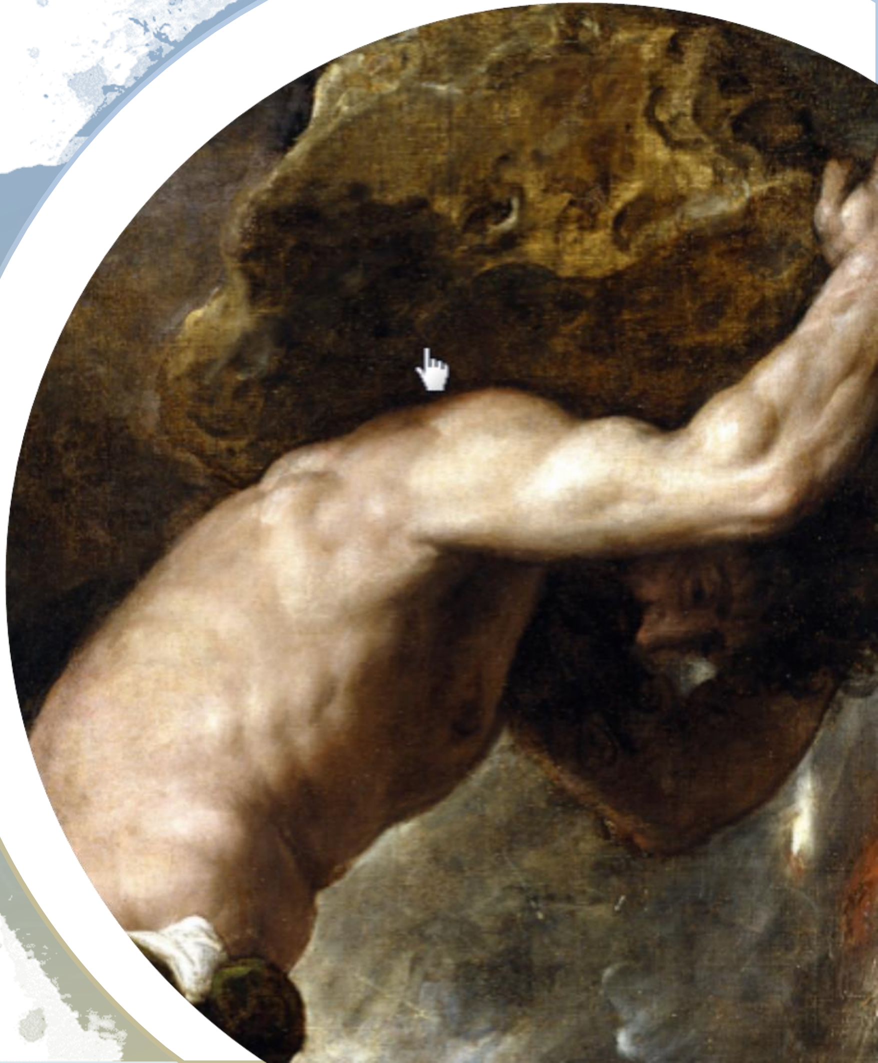
ATD Checklist
- Categorization
RQ 2.1 - Debt
Dependency violations 18
Non-Uniformity of patterns and policies 7
Code issues 7
Inter-dependent resources 6
Lack of mechanisms for addressing Non Functional Requirements 5

ATD Impediments
- Challenges
- Negative effects
RQ 2.2
RQ 2.2 - Interest
Detection 12
Time perspective 9
Complexity 6
Flexibility 3
Maintenance and evolvability 16
Innovation and system growth 4
Performance degradations 2
Reliability 4

ATD Management
- ATDM Activities
- Methods/Tools
- Refactoring
RQ 3.1
RQ 3.3
RQ 3.3 - Principal
ATDM Process 3
Identification
Measurement
Prioritization
Repayment
Monitoring
Measuring 7
Tracking 8
Evaluating 7
Extent none/ partial/full) 4
Timing 9
Resources 4
Cost- Benefit 9

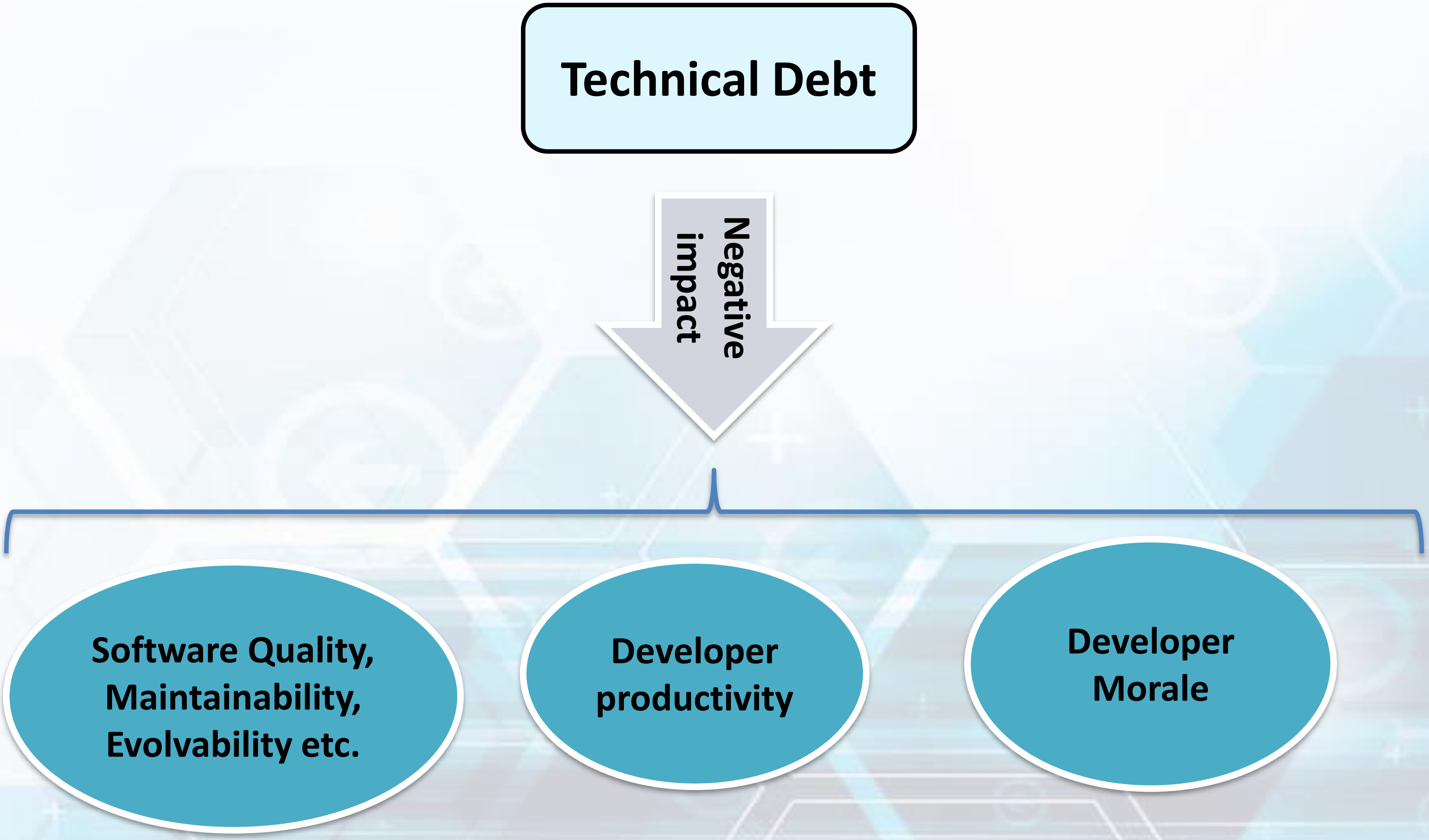
- - → = aspects linked to Complexity
····→ = aspects linked to Maintenance and Evolvability
→ = other generic relations between aspects
□ = indication of publications fitting each aspect

[T. Besker, A. Martini, and J. Bosch, “Managing architectural technical debt: A unified model and systematic literature review,” Journal of Systems and Software, vol. 135, no. Supplement C, pp. 1-16, 2018/01/01/, 2018.



Technical Debt: always negative?

Technical Debt – so far



Technical Debt

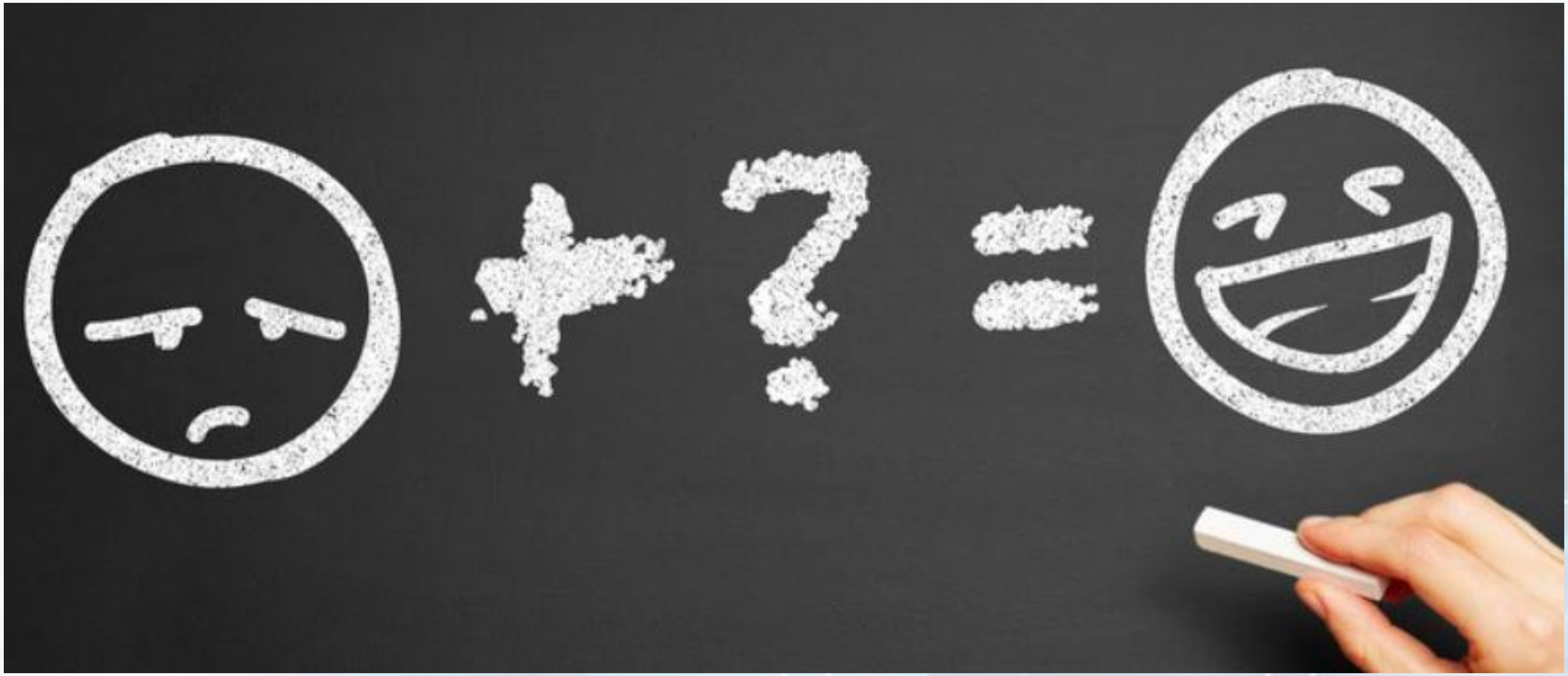
Negative impact

Software Quality, Maintainability, Evolvability etc.

Developer productivity

Developer Morale

Any suggestions when taking on TD can be beneficial?



Is it always a bad thing to take on Technical Debt?

- It’s about making informed decisions and be aware of the consequences
- Depends on the amount of the interest cost
- Depends on the variance on the interest (growing or stable)
- Possible spending the money and time on new features that can generate even more value to the company instead of paying back the debt (called refactoring)

Software Startups

Startups contra Mature Software companies

Startup Companies

Software development in **Startups**:

- Freshly created company, no history
- Main goal is to grow their business
  Extreme pressure to get to the market quickly
- Limited resources and limited budget
- High uncertainty
- Need early feedback from customers

Mature Software developing companies

Software development in **Mature** companies:

- Less pressure to get to the market quickly
- More resources
- Less uncertainty

Besker, T., et al. (2018). Embracing Technical Debt, from a Startup Company Perspective. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).

Startups and Technical Debt

- Taking on Technical Debt can be **beneficial** for Startups:

- Speed up time-to-market
- Allowing them to release their product to end-users faster
- Get feedback
- Evolve the software
- Preserve capital

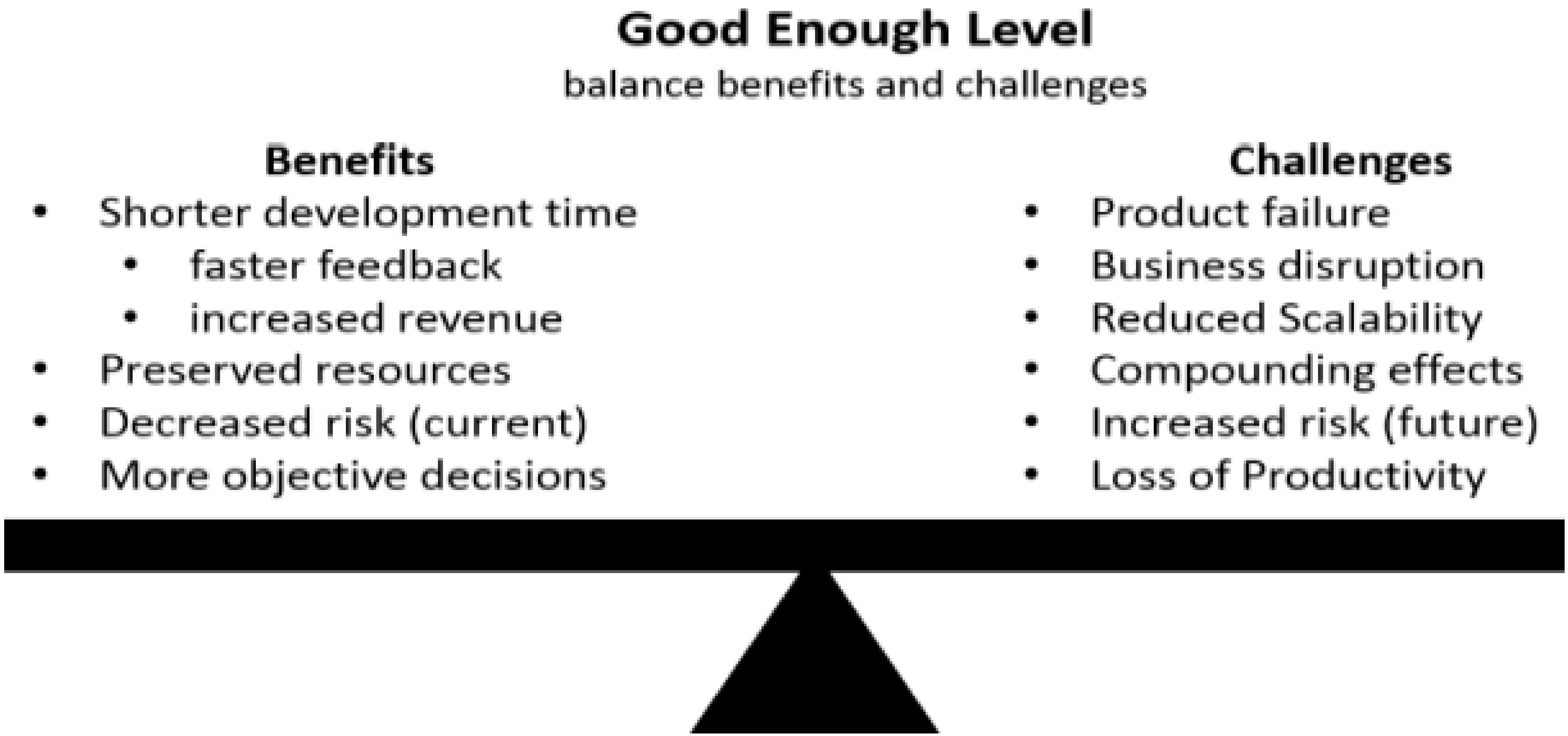
Besker, T., et al. (2018). Embracing Technical Debt, from a Startup Company Perspective. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).

Technical Debt must be managed

Unmanaged TD can have negative consequences, such as the death of the startup itself!

Besker, T., et al. (2018). Embracing Technical Debt, from a Startup Company Perspective. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).

A balance between Benefits and Challenges



Good Enough Level

balance benefits and challenges

Benefits

- Shorter development time
  - faster feedback
  - increased revenue
- Preserved resources
- Decreased risk (current)
- More objective decisions

Challenges

- Product failure
- Business disruption
- Reduced Scalability
- Compounding effects
- Increased risk (future)
- Loss of Productivity

Besker, T., et al. (2018). Embracing Technical Debt, from a Startup Company Perspective. 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).

thank you!

Terese Besker

besker@chalmers.se