

# Technical Debt Management in Visma

Mili Orucevic  
Chief Software Quality Engineer



nguages

h, Swift,

ossible

la, Dart,

gian, C#,

ish, C++,

C Sharp,

, Kotlin,

Finnish.



## Agenda

- Technical Debt in General
- How are we working with Technical Debt in Visma?
- Dive into some examples
  - Assessment
  - Dashboards
  - Indexes
  - Example of large Technical Debt
- Research: Why Technical Debt Matter?



# Mili Orucevic

## Chief Software Quality Engineer

<https://www.linkedin.com/in/milio>



# Lehman's Law of Software Evolution

## Law of Continuing Change

"A system must be continually adapted or it becomes progressively less satisfactory"

## Law of Increasing Complexity

"As a system evolves, its complexity increases unless work is done to maintain or reduce it."

# Technical Debt definition

"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. Objects make the cost of this transaction tolerable. The **danger occurs when the debt is not repaid**. Every minute spent on **not-quite-right code** counts as **interest on that debt**. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object- oriented or otherwise"

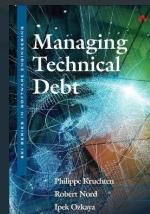
1992 Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)

**Ward Cunningham**

"In software-intensive systems, technical debt consists of design or implementation constructs that are **expedient in the short term** but that set 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, but not only, maintainability and evolvability."

Kruchten, Nord, Ozkaya (p.5)

## Managing Technical Debt



Software systems are prone to the **build up of cruft** - deficiencies in internal quality that **make it harder than it would ideally be to modify and extend** the system further.

Technical Debt is a metaphor, coined by Ward Cunningham, that frames how to think about dealing with this cruft, thinking of it like a financial debt. The extra effort that it takes to add new features is the interest paid on the debt.

Blogpost, 21.05.2019:  
<https://martinfowler.com/bliki/TechnicalDebt.html>

**Martin Fowler**

# Technical Debt Landscape

Business

New Features

Additional  
Functionality

## Architecture

Architecture smells  
Pattern Violations  
Structural Complexity

## Code

Code Complexity  
Code Smells  
Coding Style Violations  
Low Internal Quality

Visible →→

Defects

Low External Quality

End Users

## Production Infrastructure

Build, Test and Deploy Issues

## Evolvability

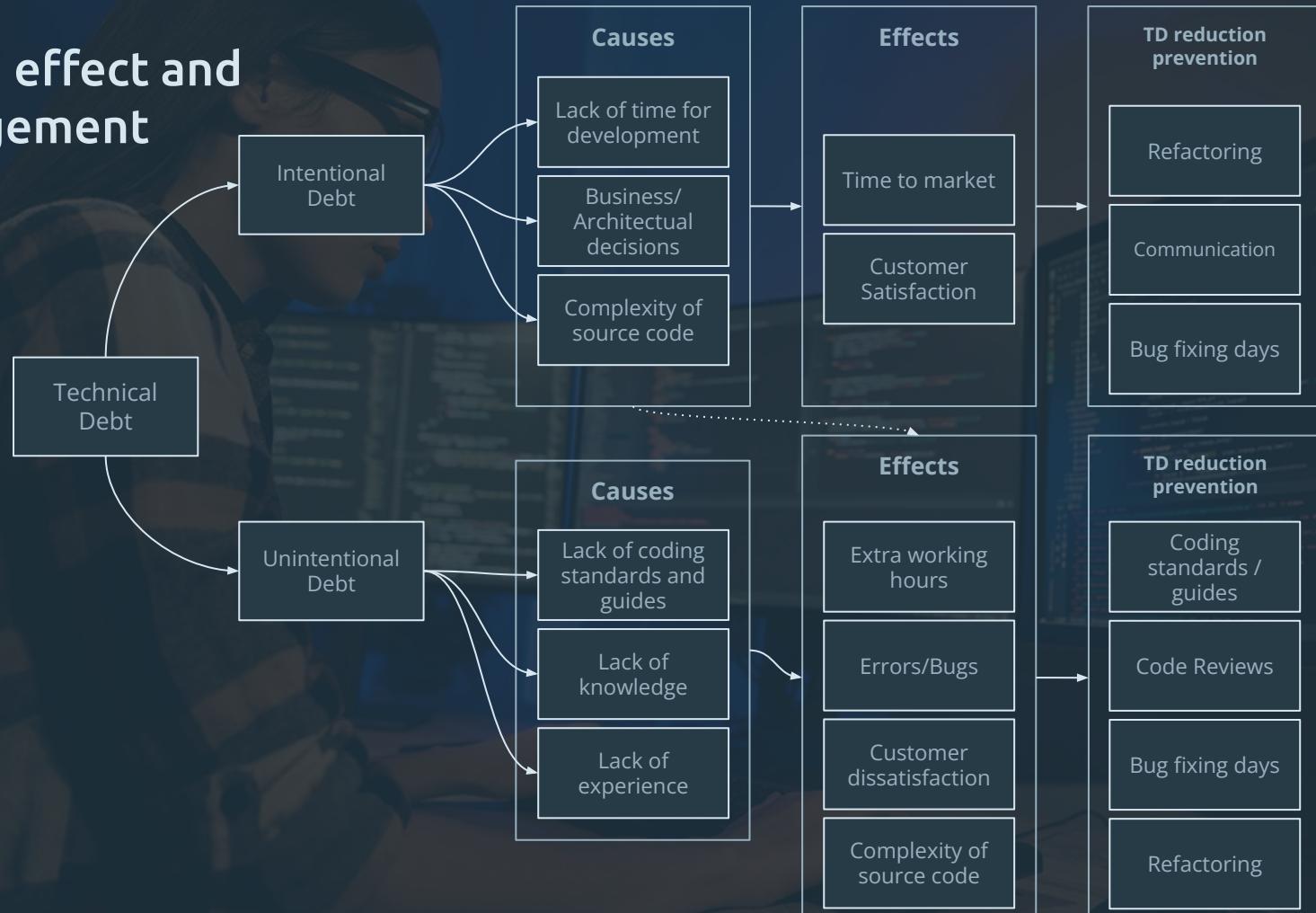
Product  
Roadmap

## Internal Team Work

## Maintainability

End Product

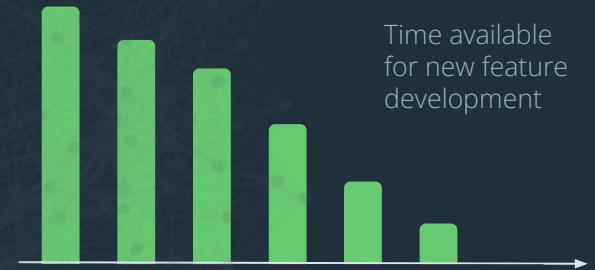
# Cause, effect and management



# Visualizing Technical Debt

Possible symptoms of Technical Debt

- Decline in number of releases
- Increase in delivery lead time
- Less and less time for developing new features
- Might treat symptoms instead of real issues



Time available  
for new feature  
development

No one is here

Won't be here for  
long



# Measuring Technical Debt

## Static Code Analysis

- 18626 days or 51 years of technical debt,  
for 1106 projects
  - ~16 days effort per project
- First 10 days of March, 51 days of effort of  
technical debt was introduced
  - 7 min per project per day

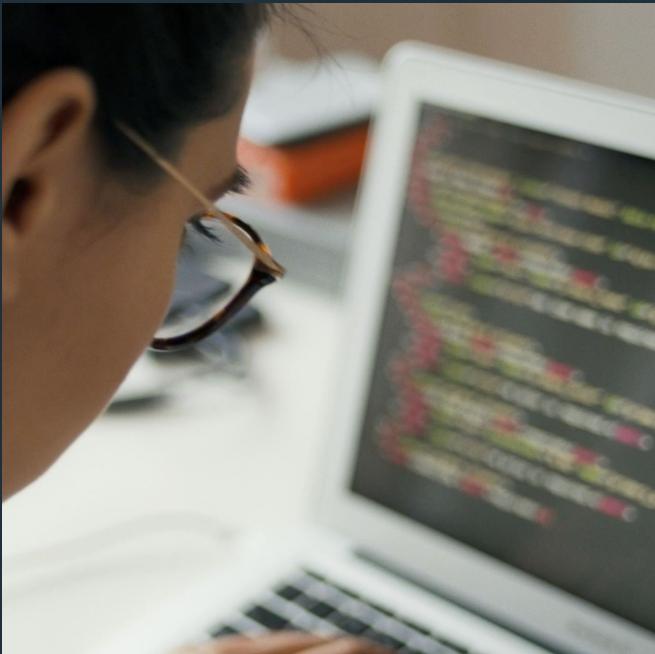
**Should not be the only way to  
measure technical debt**

## Other examples outside Visma

- Project: 15 year old code based
- Technical debt accumulated: 4000 years



How are we creating  
awareness and work with  
**Technical Debt** in Visma?





# Visma - a SaaS company

**14 000**  
Engaged employees

With more than **6 500**  
developers

Since 2014 more than  
**140 companies**  
have joined Visma

**1 135 000**  
Customers

We are **where you are**  
Strong local presence with  
more than **150**  
locations

Running through Visma's  
systems in September '21:

**10,3 million payslips**  
**23,2 million invoices**

# What did we want to solve?

Raise the **awareness** of technical debt in teams

Create **visibility** of technical debt

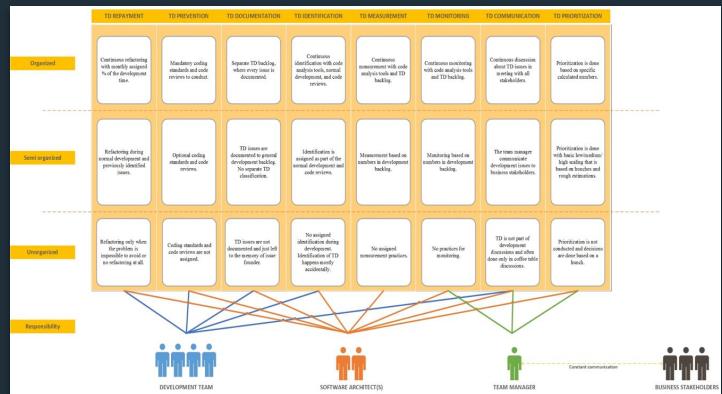
Common **process** for dealing with technical debt

Keep a technical debt **register** for teams

Ensure that technical debt is **planned** and **repaid** where reasonable



# Technical Debt Maturity Assessment



Technical Debt Maturity Model		Prevention	
		Primary responsible roles: Architects, developers, and anyone else writing code	
		Unorganized (level 0) <input type="checkbox"/> The team has not agreed on coding standards and does not perform code review. Semi-organized (level 1) <input type="checkbox"/> Following the team's coding standards and performing code review is optional or not yet. Organized (level 2) <input type="checkbox"/> The team follows coding standards and performs code review as an individual charge.	
TD Self-assessment		<b>TD Repayment</b>  <b>TDM activity / TDM levels</b>	
		Organized (level 3) <input checked="" type="checkbox"/> Continuous refactoring with monthly assigned of the development backlog.	
		Identification Primary responsible roles: Architects, developers, and anyone else that witnesses technical debt	
		Unorganized (level 0) <input type="checkbox"/> The team does not identify technical debt during the development process, except when it happens accidentally. Semi-organized (level 1) <input type="checkbox"/> The team identifies technical debt during development and code review. Organized (level 2) <input type="checkbox"/> The team identifies technical debt during development and code review. <input type="checkbox"/> The team identifies technical debt using static code analysis tools (e.g. Refactor, SonarQube, Coverity). <input type="checkbox"/> The team identifies technical debt as part of a separate process like incident review and problem management.	
		Documentation Primary responsible roles: Architects and developers	
		Unorganized (level 0) <input type="checkbox"/> The team does not document any technical debt or documents it in a separate backlog. Organized (level 1) <input type="checkbox"/> Technical debt is documented in the same backlog as everything else.	
		Analysis Primary responsible roles: Architects and developers	
		Unorganized (level 0) <input type="checkbox"/> Refactoring only when the problem is impossible to avoid or no refactoring at all. Organized (level 1) <input type="checkbox"/> Development team, software architect(s)	
		Monitoring Primary responsible roles: Product/Service Owner and Architects	
		Unorganized (level 0) <input type="checkbox"/> Technical debt is not monitored. Semi-organized (level 1) <input type="checkbox"/> Technical debt is measured over time in JIRA by looking at the number of technical debt issues. Organized (level 2) <input type="checkbox"/> Different types of technical debt metrics is measured over time by looking at the sum of all severity scores and metrics in SonarQube.	
		Communication Primary responsible roles: Product/Service Owner, architects and developers	
		Unorganized (level 0) <input type="checkbox"/> The team rarely talks about technical debt internally (in the team) or externally (with stakeholders). Semi-organized (level 1) <input type="checkbox"/> The team frequently talks about technical debt internally (in the team). Organized (level 2) <input type="checkbox"/> The team frequently communicate technical debt risks to stakeholders, focusing on the business and/or customer value of staying away from technical debt with the highest priority. The team also communicates technical debt risks to business stakeholders, focusing on the business and/or customer value (with the severity score as the key of doing this).	
		Planning Primary responsible roles: Product/Service Owner and architects	
		Unorganized (level 0) <input type="checkbox"/> The team does not have a well-defined process for deciding what technical debt should be paid down next. Semi-organized (level 1) <input type="checkbox"/> The team decides on what technical debt should be paid down next based on the value of the Priority field in JIRA. Organized (level 2) <input type="checkbox"/> The team decides on what technical debt should be paid down next based on the priority score.	
		Repayment Primary responsible roles: Product/Service Owner	
		Unorganized (level 0) <input type="checkbox"/> The team only pays down technical debt when it becomes blocking issue. Semi-organized (level 1) <input type="checkbox"/> The team performs sufficient refactoring as part of the normal backlog prioritization and development process. Organized (level 2) <input type="checkbox"/> Some of the team's capacity is leveled meant for paying down technical debt (20% or more) if the industry standard and highly recommended. Superorganized (level 4) <input type="checkbox"/> The team has instituted a maximum cap on technical debt, and will dedicate 20% of their capacity to paying down their non-debt whenever the agreed cap is exceeded.	
		TD Communication	
		Continuous communication about TD issues in meeting with all stakeholders.	
		TD Prioritization	
		<input checked="" type="checkbox"/> Prioritization is done based on specific calculated numbers. Recommended to use Severity as the calculated number.	

Inspired by research done at the Lappeenranta University of Technology by Jesse Yli-Huumo.

# Technical Debt Maturity Assessment

## Latest version

Category	Requirements
Prevention	<input type="checkbox"/> The team follows coding standards and performs code review for all non-trivial changes <input type="checkbox"/> The intention of implementing any workaround or corner-cutting is discussed within the team before any code is written; team decides together if the workaround will be implemented or more time will be allocated to avoid it > More Info
Identification	<input type="checkbox"/> The team identifies technical debt during implementation and code reviews <input type="checkbox"/> The team identifies technical debt using static/dynamic code analysis tools (e.g. ReSharper, SonarQube, Coverity, Snyk) and adds it in the backlog. > More Info <input type="checkbox"/> The team identifies technical debt as part of operational processes like incident reviews and problem management <input type="checkbox"/> A process to identify outdated dependencies is implemented (e.g Dependabot, dotnet-outdated)
Documentation	<input type="checkbox"/> Technical debt is documented in the same backlog as everything else and the NFR label is used <input type="checkbox"/> The team documents tech debt about to be introduced (intentional) technical debt > More Info <input type="checkbox"/> Our top 3 TD issues are logged and documented in a way that are understandable for stakeholders and decision makers as well. Business impact, if relevant, is part of the documentation. > More Info
Analysis	<input type="checkbox"/> Technical debt is analyzed and a severity score is calculated and documented in JIRA > Why Severity Scores? <input type="checkbox"/> The most important technical debt for our product is reflected in the top 3 TD issues logged in Jira
Monitoring	<input type="checkbox"/> All types of technical debt is visible in Jira by looking at the sum of severity scores. Here are suggestions for TD JIRA Dashboard / TD Dashboard > Click here to expand...
Communication	<input type="checkbox"/> The team continuously communicates technical debt risks to stakeholders and decision makers, focusing on the business and/or customer value of paying down the technical debt with the highest severity score. (Using the ArchTech Maturity Index and a list of technical debt issues in JIRA sorted by the severity score are two ways of doing this.) > Why is it important to communicate and be transparent about technical debt?
Planning	<input type="checkbox"/> The team includes technical debt decisions and planning into their periodic processes. > Intention and More Info <input type="checkbox"/> The team decides on what technical debt should be paid down next based on the severity scores
Repayment	<input type="checkbox"/> Some of the team's capacity is always reserved for paying down technical debt (20% or more is the industry standard and highly recommended)

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Technical Debt Maturity Assessment

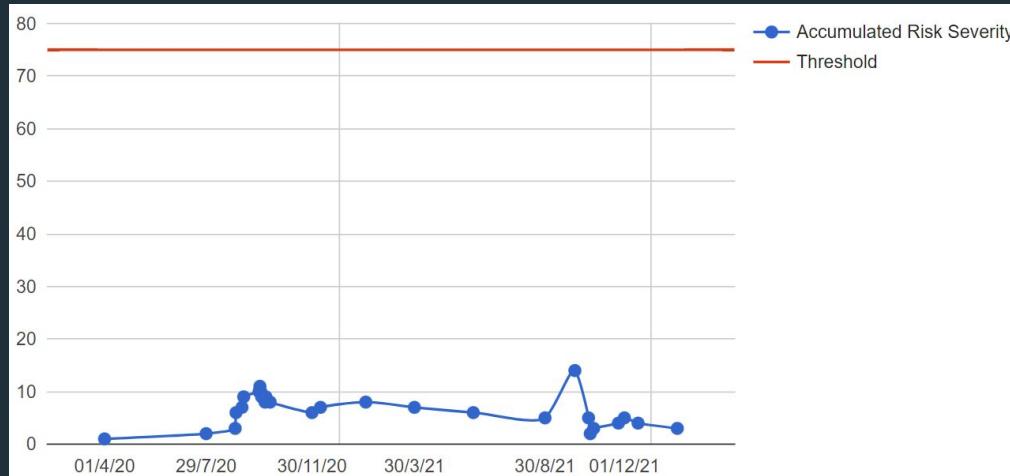
Category	Description
<b>Prevention</b>	How is introduction of unintentional technical debt prevented?
<b>Identification</b>	How, and where is technical debt identified?
<b>Documentation</b>	How is technical debt documented and labeled?
<b>Analysis</b>	How is technical debt analyzed with emphasis on risk likelihood, impact and severity?
<b>Monitoring</b>	How is technical debt monitored over time, f.ex. trends, top TD issues
<b>Communication</b>	How is technical debt communicated outside the team, f.ex. towards stakeholders?
<b>Planning</b>	How is technical debt included and prioritized in the improvement plans?
<b>Repayment</b>	How is technical debt repaid?

# Example assessment

[Link to assessment](#)

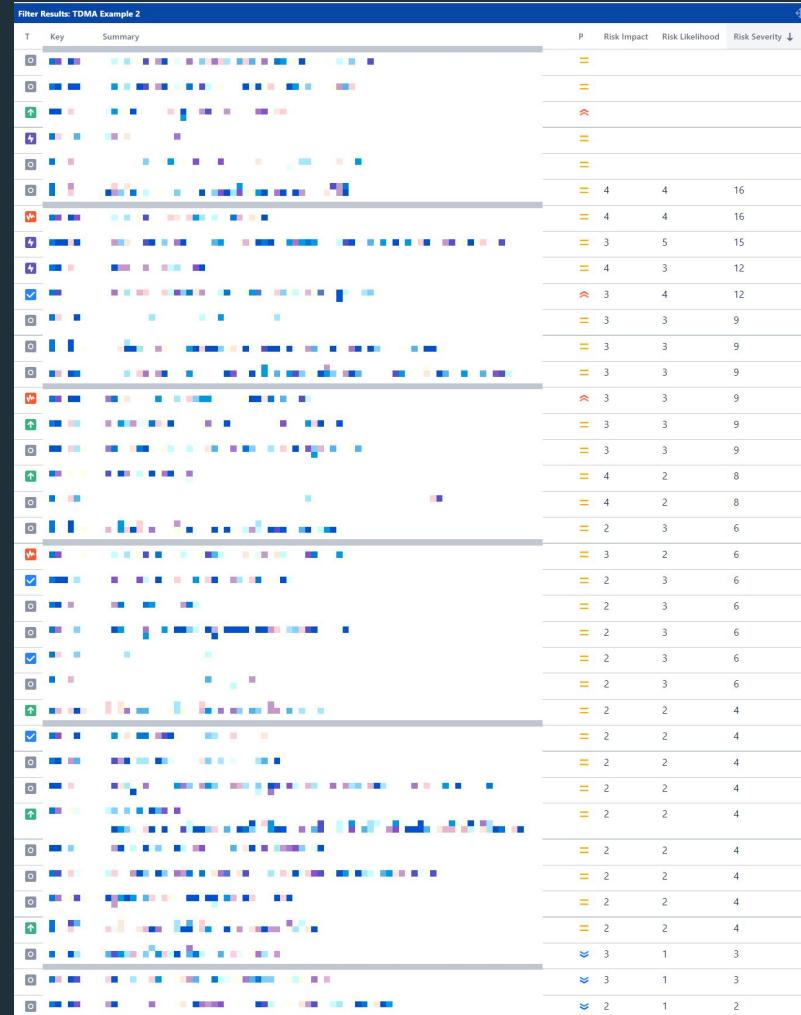


# Dashboard - team A



Filter Results: TDMA Example			
T	P	Risk Impact	Risk Likelihood
✓	=	3	9
✓	=	3	1
○	=	1	1
○	=	1	1
✓	=	1	1
○	▼	1	1
○	▼	1	1
✓	=	1	1
✓	=	1	1
↑	=	1	1
✓	=	1	1
1-12 of 12			

# Dashboard - team B



# Examples from SonarQube

The screenshot displays the SonarQube interface with three main sections: Quality Gate Status, Measures, and Conditions.

**Quality Gate Status:**

- Passed:** All conditions passed.
- Failed:** 1 condition failed. On New Code: 1 New Critical Issues is greater than 0.
- Failed:** 1 conditions failed. On New Code: 11 Bugs, 4 Vulnerabilities, 0 Security Hotspots.

**MEASURES:**

- New Code: Since November 10, 2020
- Overall Code

**Conditions:**

**Conditions on New Code**

Conditions on New Code apply to all branches and to Pull Requests.

Metric	Operator	Value
Coverage	is less than	60.0%
Duplicated Lines (%)	is greater than	3.0%

**Conditions on Overall Code**

Conditions on Overall Code apply to branches only.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	2.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Rating	is worse than	A
Unit Test Errors	is greater than	0
Unit Test Failures	is greater than	0

**Measures Summary:**

- Debt: 13d
- Code Smells: 340
- Maintainability: A

**Metrics at the Bottom:**

- Coverage: 77.7% (Coverage on 17k Lines to cover)
- Unit Tests: 994
- Duplications: 2.0% (Duplications on 41k Lines)
- Duplicated Blocks: 53

# Examples from SonarQube

The screenshot displays the SonarQube interface, specifically the code review and analysis section. On the left, a vertical sidebar lists several code smells:

- 'System.Exception' should not be thrown by user code. (Code Smell)
- 'System.Exception' should not be thrown by user code. (Code Smell)
- Refactor this code to not nest more than 3 control flow statements. (Code Smell)
- 'System.Exception' should not be thrown by user code. (Code Smell)
- Remove the unused local variable 'request'. (Code Smell)
- Fix this implementation of 'IDisposable' to conform to the dispose pattern. (Code Smell +2)

A message at the bottom of the sidebar states: "Unused local variables should be removed". Below this, a summary says "Unused local variables should be removed" and provides details: "Code Smell Minor unused Available Since Aug 14, 2015 SonarQube (C#) Constant/Issue: 5min".

The main area shows a snippet of C# code with annotations:

```
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
private async Task<...> GetSecretValue(string secretName)  
{  
    var request = new GetSecretValueRequest { SecretId = secretName };
```

A callout box highlights the line "var request = new GetSecretValueRequest { SecretId = secretName };" with the message: "Remove the unused local variable 'request'. Why is this an issue?". It includes a "last year ▾ L73 %" badge and a "unused" badge.

At the bottom right, two footer messages are visible: "General exceptions should never be thrown" and "Utility classes should not".

# “Not our fault”

## Example of a bigger technical debt

- Unintentional
- External factors
- Time

“Move from AngularJS to new version of Angular”

~ 1583 hours = 66 days

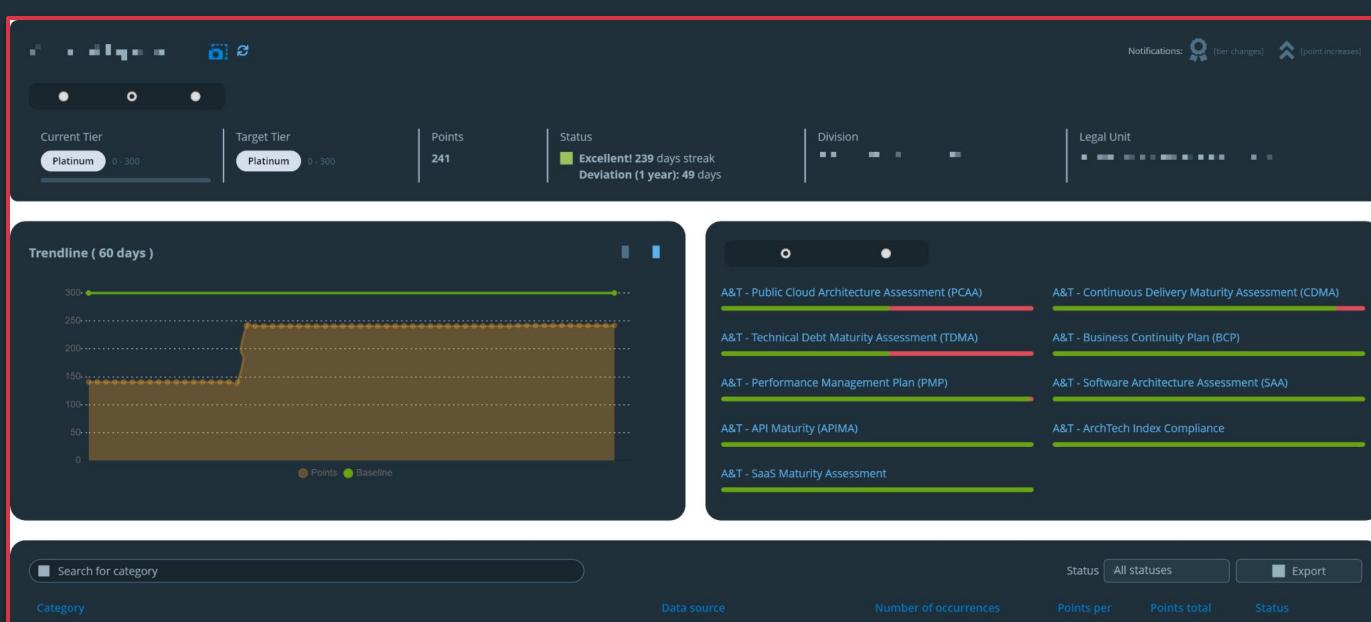


# Automatically track deviations

Visma Index



# Visma Index



# Visma Index



# Why Technical Debt Matter?

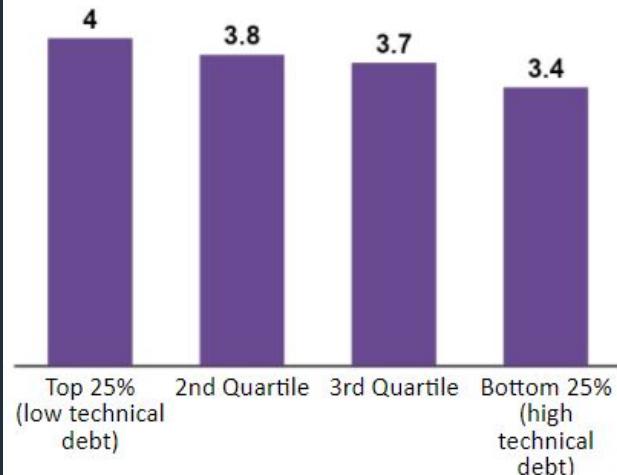
Lessons learned from own research



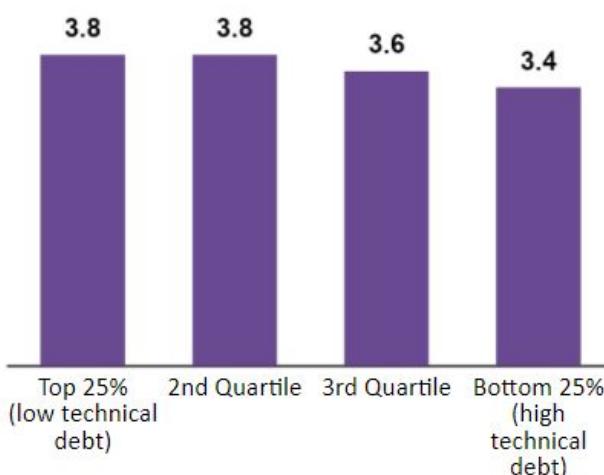
```
140         target="_blank"
141         rel="noopener noreferrer"
142         href={trackUrl}
143       >
144       Instagram
145     </a>
146   </li>
147 </ul>
148 </div>
149   );
150 }
151
152 renderWhatShowLinks() {
153   return (
154     <div className={styles.showLinks}>
155       <h4 className={styles.showLinksTitle}>
156         {this.renderWhatShowLinksTitle()}
157       </h4>
158       {this.renderWhatShowLinksList()}
159     </div>
160   );
161 }
162
163 renderWhatShowItem(title, url) {
164   return (
165     <li className={styles.footer}>
166       <a href={trackUrl(url)}
167         target="_blank"
168         rel="noopener noreferrer"
169       >
170         {title}
171       </a>
172     </li>
173   );
174 }
175
176 renderFooterSub() {
177   return (
178     <div className={styles.footerSub}>
179       <a href="/" title="Home - Unsplash">
180         <img
181           type="logo"
182           className={styles.footerSubLogo}
183         />
184       </a>
185       <span className={styles.footerSlogan}>
186         <img alt="Unsplash logo" />
187       </span>
188     </div>
189   );
190 }
191
192 render() {
193   return (
194     <footer className={styles.footerGlobal}>
195       <div className="container">
196         {this.renderFooterMain()}
197         {this.renderFooterSub()}
198       </div>
199     </footer>
200   );
201 }
```

# Technical debt matters!

Customer-centricity vs. Technical Debt (quartiles)



Innovation vs. Technical Debt (quartiles)



Products with **more technical debt** are developed by teams with **lower customer-centricity and innovation scores**.

Presumably, it's difficult to have time and energy for being innovative and customer-centric when you have work with and around old technology and practices.

# Technical debt matters!

The reduced innovation and customer-centricity in products with high technical debt drives down customer satisfaction and revenue growth.



1. In general, products with less technical debt are more successful
  - Higher product NPS
  - Higher product growth rate

2. Product lines in the bottom 25% of technical debt have higher growth rate than expected. These product lines tend to have lower CBV, and could be smaller products that have accumulated technical debt through rapid growth.



Entrepreneurial  
Responsible  
Dedicated  
Inclusive

---

Make progress happen

