TECHNICAL-, SOCIAL- AND PROCESS DEBT IN LARGE-SCALE AGILE
AN EXPLORATORY CASE-STUDY
Motivation: (Technical) Debt is responsible for development crises in large-Agile organizations

How do teams coordinate about debt?

- Teams need to catch the debt timely to avoid paying too much "interest"
Goal of our study

- Understand how teams *elicit* and *discuss* different types of debt in large-scale Agile

  - At *team* level
  - At *inter-team* level
  - How much do they need to *coordinate*?
Technical, Social and Process Debt

- Technical Debt is **heavily studied** and it’s of high interest for software companies
  - TechDebt @ ICSE
  - SEaTeD @ SEAA
  - Practical and research talks @ XP
  - ...

- Social Debt has been defined but it’s not well studied

- Process Debt has been mentioned in a few papers, but **not even defined**
Current Definition of Technical Debt

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.

Current Definition of Technical Debt

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.

Social Debt

“the presence of sub-optimality in the development community, which causes a negative effect” *

Process Debt

- There is no definition, although it’s mentioned in a few papers.

- We define it, deriving from the previous two:
  - “a sub-optimal activity or process that might have short-term benefits, but generates a negative impact in the medium-long term”

Antonio Martini - PhD in Software Engineering
Research Methodology: Retrospectives

Debt issue

Process Debt
Social Debt
Technical Debt
The study of retrospectives

Retrospective Team A
Retrospective Team B
Retrospective Large Scale

Time
The study of retrospectives

Team only

Retrospective Team A

Retrospective Team B

Retrospective Large Scale

Team only

SD

TD

PD

TD

PD

TD

PD

SD

SD

SD

PD

TD

PD

TD

Time
The study of retrospectives

Antonio Martini - PhD in Software Engineering
The study of retrospectives

Retrospective
Team A

Retrospective
Team B

Re-proposed at inter-team level

Retrospective
Large Scale

time
# Results: kind of debt found

<table>
<thead>
<tr>
<th>Debt Type</th>
<th>Team level</th>
<th>Also Inter-team level</th>
<th>~ % reported to inter-team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>26</td>
<td>9 (+ 2 only)</td>
<td>0.35</td>
</tr>
<tr>
<td>Social</td>
<td>37</td>
<td>15</td>
<td>0.4</td>
</tr>
<tr>
<td>Process</td>
<td>24</td>
<td>17</td>
<td>0.7</td>
</tr>
</tbody>
</table>

## Examples

<table>
<thead>
<tr>
<th>Debt Type</th>
<th>Team level</th>
<th>Also Inter-team</th>
<th>Inter-team only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical</td>
<td>Comp. Deployment issues</td>
<td>Unstable API</td>
<td>Lack of End-to-End tests</td>
</tr>
<tr>
<td>Social</td>
<td>Lack of UX competence in the team</td>
<td>Team autonomy</td>
<td>-</td>
</tr>
<tr>
<td>Process</td>
<td>Team planning</td>
<td>More demo sessions</td>
<td>-</td>
</tr>
</tbody>
</table>
Conclusions

- **Process Debt**
  - Many issues reproposed at inter-team level (~ 0.7)
  - Need for inter-team coordination

- **Social Debt**
  - Heavily discussed at team level (more than the other kinds)
  - Some issues reproposed (~ 0.4)
  - Need for selected inter-team coordination

- **Technical Debt**
  - Discussed at team level as much as process debt
  - Some tests issues discussed only at inter-team
  - Some issues reproposed (~ 0.35)
  - Need for selected inter-team coordination
  - No Code Debt
    - Use of other technical fora? Don’t need coordination? Open question

- **Debt needs inter-team coordination** to be managed
Limitation and future work

- Only 2 teams and 1 large-scale retro analyzed
  - We have planned more studies

- Only retrospectives analyzed
  - Analyze other data sources
    - E.g. Confluence, slack, arch. chapters, etc.
Don’t be too busy to discuss Debt!

Questions? Comments?
Extra example
Optimal architectural decision

Example:
- Standard public API

Let's put a standard API here... so later we can update the component independently.
But with fast delivery comes...

- Deliver fast!

We need these new features! Our competitor is already delivering them!

We have to deliver fast, let's use the private API... we’ll change it later.
...the accumulation of sub-optimal decisions...

- The violation is spreading to many components

- We have to deliver fast, let’s use the private API!
  We’ll change it later...
...until, one day...

- New requirement

Ok, we can replace this component since we have a standard API!

We need these new features! Our competitor is already delivering them!
...the development is not fast anymore...

Costly to remove the violation and difficult to estimate the impact

Standard API

Private API (ATD)

Comp A

Comp B

Comp C

Comp D

Comp E

OH NO! We have to change everything!

We need these new features! Our competitor is already delivering them!
...and a crisis starts.

So should we refactor or continuing with other features?

We have to refactor, but we need time...

Impossible to refactor now! We need to deliver the features!
So what is Architectural Technical Debt?

- **Non-allowed dependencies**
  - Save time by non-applying the optimal solution

- **Cost of removing dependencies**
  - How much does it cost to provide the optimal solution?

- **Extra evolution cost**
  - Replacing the component

- **Other impacts**
  - Increasing principal
  - Difficult/wrong estimation
  - Lead time increases

= “Taking” the Debt

= Principal

= Interest
So what is Architectural Technical Debt?

- **Non-allowed** dependencies
  - Save time by non-applying the optimal solution

- **Cost of removing** dependencies
  - How much does it cost to provide the optimal solution?

- **Extra evolution cost**
  - Replacing the component

- **Other impacts**
  - Increasing principal
  - Difficult/Wrong estimation
  - Lead time increases

= “Taking” the Debt

= Principal

= Interest

Important
What is social debt here?

- Lack of communication among the teams on the API
- Problematic communication between Management and teams on prioritization
- ...
What is process debt here?

- Lack of refactoring practice to remove the implicit dependency
- Lack of activity to assess current Technical Debt in the system