Architectural Technical Debt: A Grounded Theory

Companion material

Roberto Verdecchia[1], Philippe Kruchten[2], and Patricia Lago[1]

[1] Vrije Universiteit Amsterdam, The Netherlands
r.verdecchia@vu.nl, p.lago@vu.nl
[2] University of British Columbia, Vancouver, Canada
pbk@ece.ubc.ca

1 Content

In this companion material, we report two marginal categories related to *human factors* [1] which emerged in our theory, namely *person* and *communication*.

2 Person

This category deals with concepts related to the human nature of software professionals. As can be deduced from *symptoms* and *prioritization strategies*, people can support the discovery and prioritization of ATD items, and are ultimately at the origin and resolution of many of them. The following reports on the key concepts related to this category.

Awareness. To be able to manage ATD one must first be aware of its presence in a software-intensive system. Sharing knowledge about ATD items, their magnitude, causes, and consequences, enables gaining a common understanding of the ATD presence, leading to finer-grained strategies to cope with it. P4 describes:
"What important is the culture of knowing about the debt. We have to be extremely conscious about it. Every developer has to be aware about "I am incurring in debt now, I will have to pay this at some point". And this is a very good example of a developer who is aware of it." P4, Chief Technology Officer [PE-Q1]

Personal drive. Participants often reported *"the personal drive of individuals"* being at the origin of the identification, management, and resolution of ATD items. People championing for a certain ATD item are usually the ones who are affected by it on a daily basis, and actively advocate for its resolution. One of such occurrences is described by P6:
"It comes down to socializing it [ATD item]. You have to be [an] advocate for it. Bring it up in group meetings and one-on-one with certain people. Make sure that they absorb it, and hold it in the same kind of severity that you do." P6, Senior Software Engineer [PE-Q2]

Morale. ATD can have a deep negative effect on developer morale. Due to the encompassing and complex nature which ATD entails, the debt caused by ATD

items can affect development activities over a prolonged period of time, leading to severe consequences on the morale of developers. P2 describes:
"From a human perspective, if you wake up every day and you walk around in mud, are you motivated in doing it? You don't put much effort in it. Which then spirals in not getting much done... and then ends up with people leaving..." P2, Software Staff Engineer [PE-Q3]

The detachment between personal drive and a software intensive system due to ATD described in [PE-Q3] is further detailed by P15:
"The people that left before, were just able to cope with the debt, clock out after work, and dealt with the problems the next day without much thought." P15, Chief Software Architect [PE-Q4]

Seniority and skill set. Seniority and skill set can play a decisive role in ATD related phenomena. On one side, lack of necessary skill sets can lead to the introduction of ATD, due to a lack of fitted resources to address properly an instance at hand. P14 recalls:
"We had experience in monolithic applications, and that's the key reasons why we stayed with this gigantic code base. I wish we could have evaluated other options, but back then nobody in our team had the experience...it's a bit of a pain right now. P14, Senior R&D Manager [PE-Q5]

On the other hand, seniority and adequate skill sets are crucial in order to solve complex ATD items. Participants often described seniority as a decisive factor to address ATD for two main reasons: (i) senior developer are able to gain a better "holistic" view of software-intensive systems, and (ii) junior developer refrain from addressing ATD, due to the magnitude and resonance of changes carried out at the architectural level. P5 explains:
"Junior people don't want to change the architecture. Few people are confident enough to do so, there is a difference between imagining a change and pulling it off, a lot of people shy away from it." P5, Senior Software Engineer [PE-Q6]

Intuition. As described in [S-Q2], [S-Q6] and [PR-Q1], intuition and "gut feelings" can affect ATD by enabling to identify and prioritize ATD items. Additionally, personal intuition is also referenced by our participants as playing a role during the evaluation of the root causes, consequences, and magnitude of ATD items. P7 describes:
"It's a discussion about the gut feeling of how big something is. We don't have any story points associated with them, any well-defined number, it's just based on what each of us knows, what the problems entail, and how we can solve them." P7, Senior Software Engineer [PE-Q7]

Optimism bias. From our data emerged that inherent optimism of software developers and alike can deeply influence ATD. While optimism is crucial for the success of a software product, it can also constitute a cognitive bias which hinders development activities. P3 explains: *"Everything seems possible! It's just ego. This is always the problem. Think of software development, it's the art of making things possible, right? We can do it, of course we can! How long it will take is a different question... developers have to be optimist, otherwise they don't even start."* P3, Senior Director of SE [PE-Q8]

Our participants reported a wide range of cognitive biases associated to the optimism one, such as wishful thinking, self-serving bias [4], and the Dunning-Kruger effect [3]. Such biases notably lead to the emergence of the planning fallacy phenomenon [2], as described in [PE-Q8]. In addition to planning fallacies, the optimism bias and other related ones can lead also to the introduction of ATD. P6 reports: *"When we made this decision we assumed that, as our interactions were simple, they will continue to be simple. Plus, as they're all SQL databases, we assumed that they're probably pretty similar. So it's very easy to say that, as they are similar, "let's just pretend that they're all the same". And that was just a bit of optimism, but it resulted in many problems."* P6, Senior Software Engineer [PE-Q9]

3 Communication.

Communication of ATD related concepts resulted to be an emerging category in our data. Specifically, we identified 3 main related concepts described in following.

Exposition. Rising awareness among developers, managers, and the like, of the presence of ATD items results to be an important aspect steering ATD management and prioritization strategies. As described in [PE-Q2], pointing out the rise and establishment of ATD items can build a common knowledge among developer teams, leading to a comprehensive and shared viewpoint of the ATD present in a software-intensive system, which could not be established individually. P10 describes: *"Engineers get frustrated that they can't implement functionalities fast enough, so they complain and get vocal about it. This creates a situation where the architectural debt gets more awareness."* P10, Senior Software Engineer [COM-Q1]

Impediments. Related to the communication of ATD, data showed that creating awareness on the severeness of the ATD present in a software-intensive system is not always an easy task. This problem often lies in the communication between developers and management teams, potentially due to unclear consequences and symptoms associated to ATD items. Sometimes this leads to the negation of existing ATD, which can be detrimental to personal drive, and morale of developers. In this regard P8 stated: *"It resembles a Dr. Phil Show intervention. To fix a problem, you have to acknowledge that there is one."* P8, Senior Software Engineer [COM-Q2]

Blame. Incurring in ATD inadvertently, or leaving undocumented the rationale behind deliberately incurring in it, can lead to friction among people working on a software-intensive system. In fact, without a proper knowledge of the circumstances in which the debt occurred, undesirable discussions can arise, often finger-pointing individuals who incurred in the debt. P4 describes:

"People know when they are incurring in architectural debt. And if the people leave, afterwards it's a blame game on who is the culprit. Developers blame the old ones for taking bad architectural technical decisions, because they were not in their position." P4, Chief Technology Officer [COM-Q3]

References

1. Bourque, P., Fairley, R.E., IEEE Computer Society: Guide to the software engineering body of knowledge (2014)
2. Kahneman, D., Tversky, A.: Intuitive prediction: Biases and corrective procedures. Tech. rep., Cambridge University Press (1977)
3. Kruger, J., Dunning, D.: Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. Journal of personality and social psychology **77**(6), 1121 (1999)
4. Myers, D., Smith, S.: Exploring social psychology. McGraw-Hill (2015)