Technical Debt Trade-Off - Experiences from Software Startups Becoming Grownups

Orges Cico

Norwegian University of Science and Technology, Trondheim, Norway
orges.cico@ntnu.no

Abstract. Software startups are software-intensive early-stage companies that have high growth rates. Their time to market is often regarded as short and decisive in establishing their product/service success, thus leading to short-cuts in software engineering decisions. High accumulation of the technical debt at early stages has been documented from previous investigations. How startups rapidly becoming grownups perceive technical debt, make the primary goal of our study. We conducted semi-structured interviews with six technical and executive officers from five software startups, selected using purposive sampling. We identified four critical perceptions (managing, accepting, avoiding, ignoring technical debt) which permit them to make technical debt trade-offs. We also found that no one size fits all. Startups need to make deliberate educated decisions on how to use technical debt in their advantage.

1 Introduction

Facing Technical Debt (TD)[1] is becoming even more of an urgent need for many software startups [2,5,8]. Empirical evidence on how TD is perceived from software startups is still meager, and the need for empirical evidence is reported from [1]. Software startups are known to accumulate TDs via their early-stage prototyping and product development, which eventually requires the companies to pay the debt, causing initial growth hinders productivity [6]. At the point in time when startups shift to an established stage in term of finance and resources, the management of such TDs becomes significance from managerial perspective. Compared to previous efforts studying TDs at different startup phases, the understanding of TD management at such transitions is very limited. We aimed at understanding effective approaches for managing TDs for startups in the scaling transitions. As the first step, we formulated the following research question:
RQ: *How is Technical Debt perceived in Software Startups becoming Grownups?*

[1] Metaphoric concept of TD has been first introduced by Ward Cunningham [4] in 1992.

To answer the RQ, we designed a survey-based semi-structured interview, conducted with six Chief Executive Officer (CEOs) and Chief Technical Officers (CTOs) from five software startups, selected using purposive sampling. We focused on those startups that are almost or have already made a successful transition towards becoming Grownups[2]. Aligned to previous studies findings, we also noticed that TD is deliberately embraced as long as product/service delivery deadlines and good enough quality are met. Furthermore, we found that a TD trade-off is required in the transition from early stages to grownup stages. Eventually, we identified (1) Managing TD, (2) Accepting TD, (3) Ignoring TD, (4) Avoiding TD are the main approaches perceived from TS to achieve the TD trade-offs. Providing empirical evidence on how transitioning startups have been able to conduct a smooth transition from Minimum Viable Products (MVPs) towards qualitative product/services can help future practitioners and entrepreneurs make educated decisions.

2 Research Methodology

We aim to understand the perception of technical debt in Software Startups becoming Grownups. Therefore, the research question guided our investigation. Based on recommendations from Runeson [7] we devised a qualitative approach with semi-structured face-to-face interviews with six CEOs/CTOs from the five Software Startups.

2.1 Case Selection and Demographics

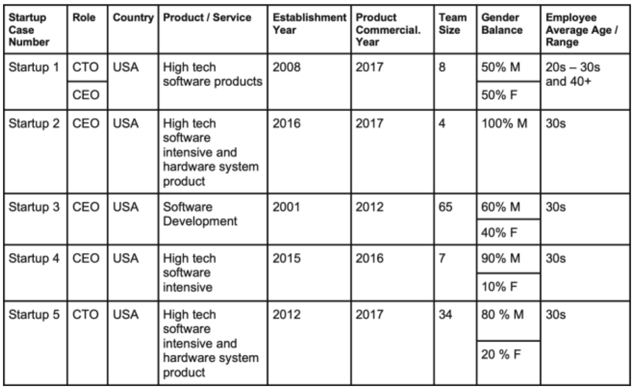
We were able to collect the sample data from a significant event where participation involved 100+ startups. The sample population has been selected using a non-probability sampling technique. We collected data from the startups' online resources after initial contact (email or face-to-face acquaintance) and then later on from CEOs and CTOs. Demographics of the five software startups are reported in Table 1.

2.2 Interview Design, Data Collection and Analysis

We performed a pilot study in constructing our interview template, which was used for later data collection from all the cases. This allowed us to focus our interview questions in connection to the RQ. The interview process took place in three parts: (1) demographic information about the startup (2) broad context on software and technological aspects of the startup (3) perception of technical debt. We interviewed six CTOs/CEOs from five different startups located in the same country and conducting geographically proximate business activities with a high tech product focus. The interviews aimed to understand the perception of

[2] Grownups are well established companies with market revenue being primary source of income.

Table 1. Software startup sample demographics.



Startup Case Number	Role	Country	Product / Service	Establishment Year	Product Commercial. Year	Team Size	Gender Balance	Employee Average Age / Range
Startup 1	CTO	USA	High tech software products	2008	2017	8	50% M	20s – 30s and 40+
	CEO						50% F	
Startup 2	CEO	USA	High tech software intensive and hardware system product	2016	2017	4	100% M	30s
Startup 3	CEO	USA	Software Development	2001	2012	65	60% M	30s
							40% F	
Startup 4	CEO	USA	High tech software intensive	2015	2016	7	90% M	30s
							10% F	
Startup 5	CTO	USA	High tech software intensive and hardware system product	2012	2017	34	80 % M	30s
							20 % F	

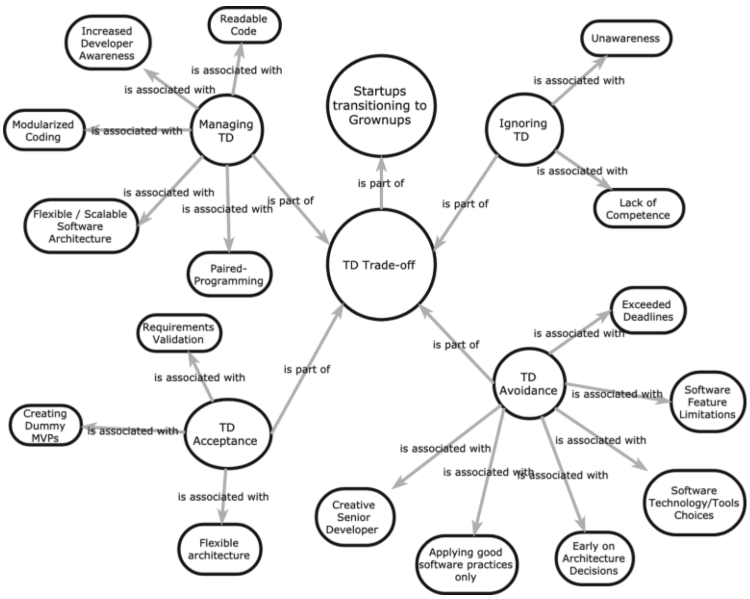
TD from startup founders, represented by both CEOs and CTOs, Table 1. After data collection, in order to obtain significant evidence, we used thematic analysis approach [3], consisting of identifying recurring patterns and themes within the interview data. The systematic analysis steps consisted in (1) **Reading the transcripts**, (2) **Coding**, (3) **Creating themes**, (4) **Labeling and connecting themes**, (5) **Drawing the results summary**, (6) **Writing results**. We also used thematic coding tools such as NVivo.

3 Results

During our analysis we created five major categories, namely TD trade-off, Managing TD, Avoiding TD, Accepting TD, and Ignoring TD, each helping to answer our RQ in the following subsections.

– **TD Trade-off.** In most cases, we noticed a repetition of the TD trade-off term. The term itself was mentioned from the interviewer, reporting positive connotation from the interviewees. This demonstrates that the perception of the TD is not totally negative or positive, but it is commonly agreed that a TD trade-off is required in the transition from early to grownup stages. For example, the CTO of Case 5 explicitly states: *"We accept TD can happen, take responsibility for it and this is all about trade-offs. Our team is highly deadline driven."*. Thus, here is where we identified different approaches (Managing TD, Accepting TD, Ignoring TD, Avoiding TD) part of TD trade-off while analyzing the perception of the CEOs/CTOs.
– **ManagingTD.** In many cases, TD management is reported as the most common option. The CEOs from two startups (Case 3, 5) emphasize the relevance the increased awareness helped them have better control over the TD. This was common even in large contingents of development teams adopting pair programming approach to software development. In Case 5, TD trade-off was

accepted, whenever deadlines had to be met. However, team members were fully aware of the situation and accepted that TD issues had to be dealt with later on. Likewise, managing and isolating code issues modules with low coupling helped in controlling TD, as reported in Case 3 (Fig. 1).



Increased Developer Awareness
Readable Code
Startups transitioning to Grownups
Unawareness
Modularized Coding
Managing TD
Ignoring TD
Flexible / Scalable Software Architecture
Paired-Programming
TD Trade-off
Lack of Competence
Requirements Validation
Exceeded Deadlines
Creating Dummy MVPs
TD Acceptance
TD Avoidance
Software Feature Limitations
Flexible architecture
Creative Senior Developer
Applying good software practices only
Early on Architecture Decisions
Software Technology/Tools Choices
is associated with
is part of

Fig. 1. Themes summary.

- **AvoidTD.** We found that avoiding TD is primarily perceived as positive when sacrificing software features seemed to be a good option. Case 3 reported that: *"We can develop anything but not everything within the given time limitations"*. However, in Case 1, avoiding TD was strongly connected with exceeded deadlines, or good software practices producing products that don't match the end-user needs. We found that early on architecture, programming language, and technology choices helped the software startups in taking precautions to avoid TD, Case 2.
- **AcceptingTD.** Although the acceptance of the TD term was a mere surprise for us, we discovered that the acting along with the TD was considered to be beneficial. Case 2, reported that requirement validation could be best achieved when introducing dummy MVPs that can be thrown away due to the large amount of TD introduced. Furthermore, the same case reports that TSs can widely accept TD if relying on easy to manipulate backend architectures. Accepting TD is perceived to be inappropriate, Case 3 reports: *"We always use best approaches, although we accept that we cannot achieve perfect software."*.

– **IgnoringTD.** We found that this category was strongly associated with a lack of TD awareness from the team, Case 3. Planning ahead to throw away prototypes can also lead to ignoring TD for those modules totally. Example made earlier with dummy MVPs from Case 2. However, to differentiate with the previous example, when a startup decides to ignore TDs they have made a deliberate long-lasting decision, which might or might not affect the product during its operational lifetime, but the reason for doing so is lack of team competence which is not possible for them to compensate.

We report **key findings:**

1. Managing TD is perceived to be an essential aspect of the TD trade-offs to be made in order to meet deadlines. Accountability for improving the software system is to be dealt with afterward. Readable code and flexible software architectures help along the process.
2. Avoiding TD can have positive as well as a negative connotation. If startups are able to cut-off features of their products, then it is recommended for them to try to avoid TDs, while applying good software development practices.
3. Accepting TD is found in two main beneficial scenarios: (1) acting along with TD to validate requirements (2) flexible backend software architectures that allow for rapid change.
4. Ignoring TD is primarily affected by lack of awareness and lack of competence.

4 Discussions

Many of the previous studies have focused on covering and addressing several startup life cycle phases by unfolding the TD challenges and benefits [2]. In our case, we focus more on a specific moment in time borderline to the transitioning from software startups to grownups. This is of significant interest because not knowing how to cope with TD at this later stage to make the big decisive jump has higher financial and technological risks. The perception of TD of succeeding startups having made the jump to grownups can be a winning and compelling choice for future ones. Another important reason for studying borderlines is also because it is there when disruptions are observed and successfully overcoming TD thresholds is required [2]. We believe that TD while transitioning to grown up company has a different perception compared to TD while at very early stage. We also provide **key recommendations:**

1. TD is going to be your best friend or best enemy, so making the right Trade-offs is crucial. No one size fits all.
2. Cut-off software features if you require less TD. This workaround can still allow software startups to meet deadlines without compromising future updates.
3. Accept TD and make it work in your advantage. Build as many dummy MVPs as possible until you are sure about requirements.

4. Hire if possible at least one highly creative senior developer. If they understand why you want to build the system, they can also tell you what you need to build.
5. Play it smart. Don't just ignore TD, because you are unaware or because you think you lack the competence. As per definition, the debt is later to be paid, unless you decide it is useful in staging your product.

5 Conclusions and Future Work

We focused on understanding how software startups transitioning in grownup, perceive TD. After interviewing five different software startups and six of the co-founders, holding either CEO or CTO roles, we identified four important perceptions (Managing TD, Avoiding TD, Ignoring TD, Accepting TD) which permit them to make TD trade-offs. We also found that no one size fits all. Startups need to make deliberate educated decisions on how to use TD in their advantage. This can only be obtained if they have a clear view of the options to cope with TD. This study provides a set of initial recommendations. We plan in the future to collect further data by surveying and interviewing larger sets of participants. The triangulation will allow us to generalize our findings and provide guidelines to be exploited by future startups.

Acknowledgement. This work was funded by the Norwegian Research Council under the project IPIT Project Number: 274816. Many thanks to Prof. Letizia Jaccheri, for the support as project leader.

References

1. Abrahamsson, P., et al.: Software startups - a research agenda. e-Informatica Softw. Eng. J **10**(1), 1–28 (2016)
2. Besker, T., et al.: Embracing technical debt, from a startup company perspective. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 415–425. IEEE (2018)
3. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qual. Res. Psychol. **3**(2), 77–101 (2006)
4. Cunningham, W.: The WyCash portfolio management system, Experience Report. In: Proceedings on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 1992) (1992)
5. Devos, N., Durieux, D., Ponsard, C.: Managing technical debt in IT start-ups-an industrial survey. In: International Conference on Software and System Engineering and Their Applications (ICSSEA) (2013)
6. Giardino, C., et al.: Software development in startup companies: the greenfield startup model. IEEE Trans. Softw. Eng. **42**(6), 585–604 (2016)
7. Runeson, P., Höost, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131 (2009)
8. Tom, E., Aurum, A.K., Vidgen, R.: An exploration of technical debt. J. Syst. Softw. **86**(6), 1498–1516 (2013)