Technical Debt Trade-off - Experiences from Software Startups becoming Grownups

No Author Given

No Institute Given

Abstract. Software startups are software-intensive early-stage companies that have higher growth rates compared to other startup breeds. 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. This study, throughout its recommendations, provides an initial road-map for future startups.

Keywords: Software Startups · Technical Debt · Software Intensive

1 Introduction

A startup is commonly defined as newly established companies with small teams, limited resource and aim for rapid scaling business models [11, 2] . At the early stage, the primary goal is to meet a marketplace need by developing a viable business model for products, services, processes, or platforms. The failure rate of startups is commonly high; however, successful startups have had a major impact on the industry [21]. This is particularly true for startups developing software-intensive products, which have shown higher rates of scaling [6], making them stand out.

Facing Technical Debt[1] (TD), is becoming even more of an urgent need for many software startups [24, 9, 3]. 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 [12]. At

[1] Metaphoric concept of TD has been first introduced by Ward Cunningham [8] in 1992. Read more on Section 2 about its relation to software startups.

the point in time when startups shifting 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?*

To address 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.

The rest of this paper is organized as follows: In Section 2, we describe the background and related work. Our startup cases context is described in Section 3.2. Our research methodology is described in Section 3. The results are presented in Section 4. Finally, we discuss the implications and limitations of our work in Section 5. Whereas, the conclusions and future work perspectives are presented in Section 6.

2 Background and Related Work

2.1 Software Startups ecosystems, development and life-cycle phases

The failure rate of startups is commonly high; however, successful startups have had a major impact on the industry [21]. Typically startups operate and evolve in an ecosystem with connections to various stakeholders, from various types of investors to incubators, accelerators and third party vendors. Startups typically undergo several development phases: Ideation (Product or Service idea), Concepting (Mission and Vision), Commitment (Team with the initial product),

[2] Grownups are well established companies with market revenue being primary source of income. Read more about startup life-cycle phases on Section 2.1

Validation (Iteration and testing the initial idea), Scaling (Focus on key performance indicators), and Establishment (Increasing growth and market potential) [7].

As shown in Figure 1, the transitions of startups from one stage to another stage can be characterized under different categories. Finance is one of the most important factors for startup survival. In the early stages, Funding is commonly based on self-contributions, in the form of self-investment (by bootstrapping between jobs) or loans (from relatives or friends). Other funding options in the early stage of startup formation can come from pre-seed or crowdfunding. In later stages, when an MVP has been developed, and iteration with the market is a must (do or die approach), the need for larger funding amounts from venture capitalists (VCs) and angel investors (AIs) becomes obvious. Finally, if the startup has developed a fully operational product or service, then the market, either local or global, decides the startups growth potential.

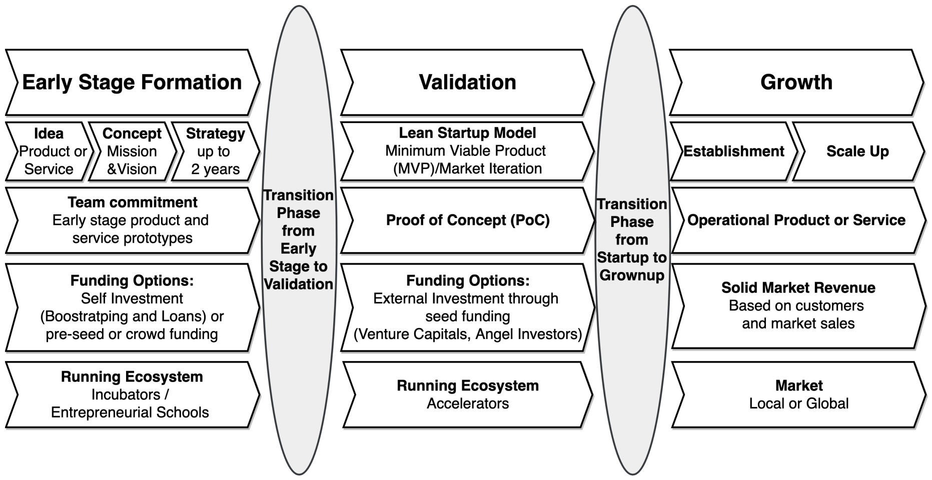

Early Stage Formation
Idea Product or Service
Concept Mission &Vision
Strategy up to 2 years
Team commitment Early stage product and service prototypes
Funding Options: Self Investment (Boostratping and Loans) or pre-seed or crowd funding
Running Ecosystem Incubators / Entrepreneurial Schools
Transition Phase from Early Stage to Validation
Validation
Lean Startup Model Minimum Viable Product (MVP)/Market Iteration
Proof of Concept (PoC)
Funding Options: External Investment through seed funding (Venture Capitals, Angel Investors)
Running Ecosystem Accelerators
Transition Phase from Startup to Grownup
Growth
Establishment
Scale Up
Operational Product or Service
Solid Market Revenue Based on customers and market sales
Market Local or Global

Fig. 1. Startup ecosystem, development and lifecycle phases

As shown in Figure 1, the transition of startups is also marked by the methodological evolution, from ad-hoc or customized development practices [17] to more principled approaches. A lean startup is a popular methodology among startups at early-stages and validation stages [20]. It focuses on shortening the product development cycle, through iterative product releases, market experimentation, and validation. Meeting the needs of early customers should reduce later failure risks of large investments. Teams that adopt a lean startup strategy develop a continuously changing MVP [19]. The MVP, comprising the technological Proof of Concept (PoC), helps identify product/service potentials with the startup limited resources. Nevertheless, if the software startups desire to transition towards the growth phase, it has to establish itself a so-call Grownup stage. The Grownup stage inherits much of the technological features, benefits, drawbacks from its successful MVP predecessor, now becoming a professionalized product/service fulfilling a specific market need.

2.2 Technical Debt

The metaphoric concept of TD has been first introduced by Ward Cunningham [8] in 1992. Becoming more of an interesting research area, further refined definitions have been provided from several future authors, such as Brown et al. [5]: *"developers sometimes accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline), and that such compromises incur a "debt": on which "interest" has to be paid and which the "principal" should be repaid at some point for the long-term health of the project."*. In early 2011, Carolyn Seaman [23] report issues associated with technical debt, and propose a technical debt management framework as well as a research plan for validation. Kruchten [15] presents the TD from a solid consolidated theory and practice standpoint. The author araises the need and requirement for more tools and methods to identify and manage TD. A practical evaluation of how TD might impact real software project throughout their lifecycle is given from Yuepu Guo et al. [14]. The authors evaluate TD effects and try to emphasize what research methodologies can be used to investigate TD management.

A systematic mapping of TD and its management has been provided from Zengyang Li [16], consolidating the TD term usage and identifying a need for more empirical studies with high-quality evidence from industry practices in managing TD.

2.3 Software Startups becoming Grownups and Technical Debt

It is only in 2013 and 2016 that the need for conducting TD research in TS context has been reported from Christophe et al. [9] and Unterkalmsteiner et al. [1]. Based on the open questions and product road-map recommendation needs to be stated by Unterkalmsteiner et al., many researchers started conducting empirical investigations in this area. Their concern has primarily been on how is technical debt introduced in different software engineering knowledge areas (code, design and architecture, environment, knowledge distribution and documentation, and testing) in TS context, identified by Tom et al[24]. For example, Gralha et al. in [13] states the following: *"Improved ability to handle technical debt results in a higher ability to prioritize requirements with greatest customer impact."*, providing us with a solid theory about dimensions (Requirement artifacts, Product quality, Knowledge management, Technical debt, Requirements-related roles) that determine product requirement practices in startups. Whereas, Max et al. [10] have made an empirical evaluation of how programming language choices might affect the level of technical debt introduced in small teams and startups. Moreover, studies reporting empirical evidence on startup product quality relevance and when TD is commonly revealed from startups in industrial setting

[9], help us make research decisions on which startup phases require more attention in managing TD. Besker1 et al. [3] also report empirical evidence on how the taking on or embracing of TD is deemed to be an essential option for TSs. The authors also analyze how the TD is accumulated and refactored at different stages of startup development to provide good enough levels of quality.

3 Research Methods

We aim to understand the perception of technical debt in Software Startups becoming Grownups. Therefore, the **RQ:** *How is Technical Debt perceived in Software Startups becoming Grownups?*, guided our investigation.

In order to gather and to interpret evidence for answering our research questions, we devised a qualitative approach. To answer it, we conducted semi-structured interviews with six CEOs/CTOs from the five TS described in detail in Section 3.2.

3.1 Case selection

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. The demographics and startup context is reported in the upcoming Section 3.2.

3.2 Case Demographics

We present in this section a brief context of the startup companies that participated in our study. 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. It is worth noting that all the interviewees are presently co-founders of the startups.

We have further summarized each startup context and their present life-cycle phase in Table 2.

3.3 Interview design

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. The first part of the survey primarily addressed demographic information about the startup. The second phase focused more in a broad context on software and technological aspects of the startup. The third part went in detail the perception of technical debt each of the samples had. Dividing the interview into different parts helped us in guiding the startups in expressing their standpoints, without being biased from our expectations. The approach followed

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	

Table 2. Startup context.

Startup Case Number	High Tech Solution	Project Management Practices	Phase	Number of Customers
Startup 1	Genomic search engine.	Agile	TG	10+
Startup 2	Solves mechanist skilled workers shortage by programming and automating them in physical component design industry. The startup is presently beyond the MVP phase and has already established a product while having a stable initial pool of customers in the market.	No particular practices.	TS transitioning to TG. MVP to Operational product is noted, with limitations.	1-5
Startup 3	Software development consultancy for high tech products, throughout pair programming practices.	In-house practices for software development complying similar Agile Practices. Heavily relying on paired programming practices.	TG.	10+
Startup 4	GPU-accelerated software for rapid secondary analysis of next-gen sequencing data.	No particular practices. Some task distribution tools	TS transitioning to TG.	5+
Startup 5	Wind turbine inspection through drone technology.	Agile practices.	TG	5+

does not compromise the data gathered since from the start, we planned a semi-structured interview having little control over the samples chosen. Although the planning of the interviews was done in collaboration between the two authors, its execution was performed by only one of them and afterward peer-reviewed again, as further discussed in Section 3.4.

3.4 Data Collection

In order to answer our RQ and based on recommendations from Runeson [22], we collected data from semi-structured, face-to-face interviews. We interviewed eight CTOs/CEOs from six different startups located in the same country and conducting geographically proximate business activities with a high tech product focus. The interviews lasted for 60 minutes and were recorded for later transcription. To facilitate the latter process, we utilized online tools (sonix.ai), delivering an approximate accuracy of around 95% in English transcriptions.

The interviews aimed to understand the perception of TD from startup founders, who are commonly represented by both CEOs and CTOs as reported in Table 1. Since both these entities represent the primary stakeholders and their interest in having a smooth transition from MVPs towards qualitative product/services while migrating in the establishment phase. This being said, we deliberately overlooked the possibility to interview also the hired developers of the companies, because they did not have any stakeholder interest. Furthermore, during the second phase of the interview, we discovered that in most cases, software engineering practice decisions were made top-down.

As explained in Section 3.3, we split our interview into three different parts, 1) demographic questions (lasting 10-15 min) 2) General software engineering practices questions (lasting 15-20 min) 3) TD perception questions. The specific questions for each part are reported in Table 3.

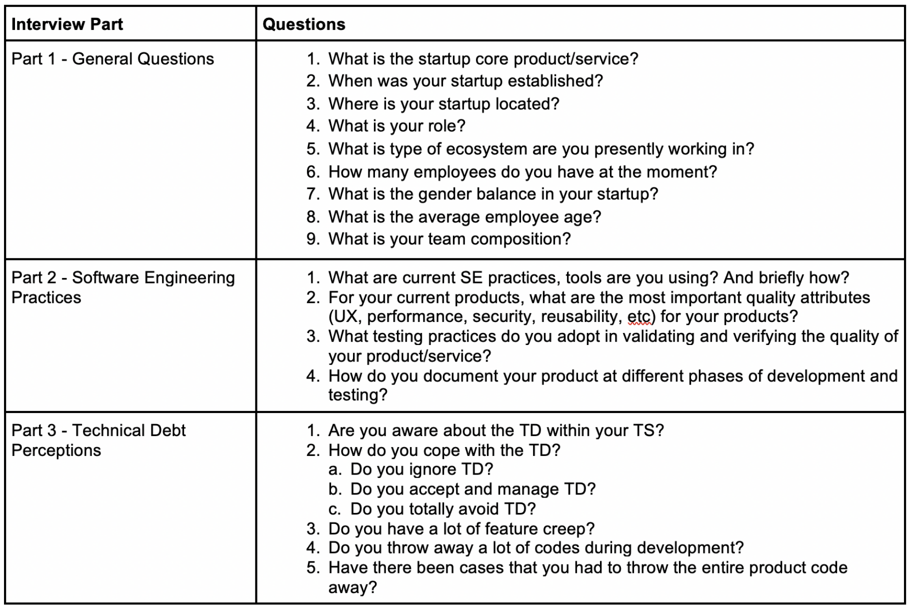
3.5 Data Analysis

After carefully collecting the data, in order to obtain significant evidence that would help us answer our RQ, we used thematic analysis approach [4], consisting of identifying recurring patterns and themes within the interview data.

The steps followed in conducting the systematic analysis consisted in:

1. **Reading the transcripts.** This step initially involved quick browsing and correction of the automatically transcribed data from the audio recordings. We made quick notes about first impressions. Later on, authors reviewed more carefully the transcribed data by reading carefully, line by line.
2. **Coding.** During this step, we focused on choosing and labeling relevant words, phrases or sentences and even larger text fragments or sections. The labels constructed reported more about opinions and perceptions related to TD. We primarily looked for repetitive and unexpected answers compared to previous theories. We tried to code as much as possible regarding the TD phenomena. To mitigate the biasing, the two authors worked separately during this coding process.

Table 3. Interview parts and questions.



Interview Part	Questions
Part 1 - General Questions	1. What is the startup core product/service? 2. When was your startup established? 3. Where is your startup located? 4. What is your role? 5. What is type of ecosystem are you presently working in? 6. How many employees do you have at the moment? 7. What is the gender balance in your startup? 8. What is the average employee age? 9. What is your team composition?
Part 2 - Software Engineering Practices	1. What are current SE practices, tools are you using? And briefly how? 2. For your current products, what are the most important quality attributes (UX, performance, security, reusability, etc) for your products? 3. What testing practices do you adopt in validating and verifying the quality of your product/service? 4. How do you document your product at different phases of development and testing?
Part 3 - Technical Debt Perceptions	1. Are you aware about the TD within your TS? 2. How do you cope with the TD? a. Do you ignore TD? b. Do you accept and manage TD? c. Do you totally avoid TD? 3. Do you have a lot of feature creep? 4. Do you throw away a lot of codes during development? 5. Have there been cases that you had to throw the entire product code away?

3. **Creating themes.** After gathering all the codes, we decided on the most relevant ones and created different categories, also defined as themes. Many initial systems from the previous step were either dropped or merged together to form new ones.
4. **Labeling and connecting themes.** During this step, we decided on which themes are more relevant and defined appropriate names for each of them. Furthermore, we also tried to identify relationships among the themes.
5. **Drawing the results summary.** After deciding over the theme importance and hierarchy, we drew a summary of the results, Figure 2.
6. **Writing results.** Our results comprised different themes reflecting the TD perceptions of the participants to the study. Based on them, we wrote the results answering our RQ in Section 4 and discussing them compared to previous studies in Section 5.

To fulfill the first five steps, we used thematic coding tools, such as NVivo 12 [18].

4 Results

During our analysis, we identified several factors that influenced how TD is perceived by the CEOs and CTOs of the startup while they are transitioning to the TG phase. Thus we created five major categories, namely TD trade-off (Section 4.1), Managing TD (Section 4.2), Avoiding TD (Section 4.3), Accepting

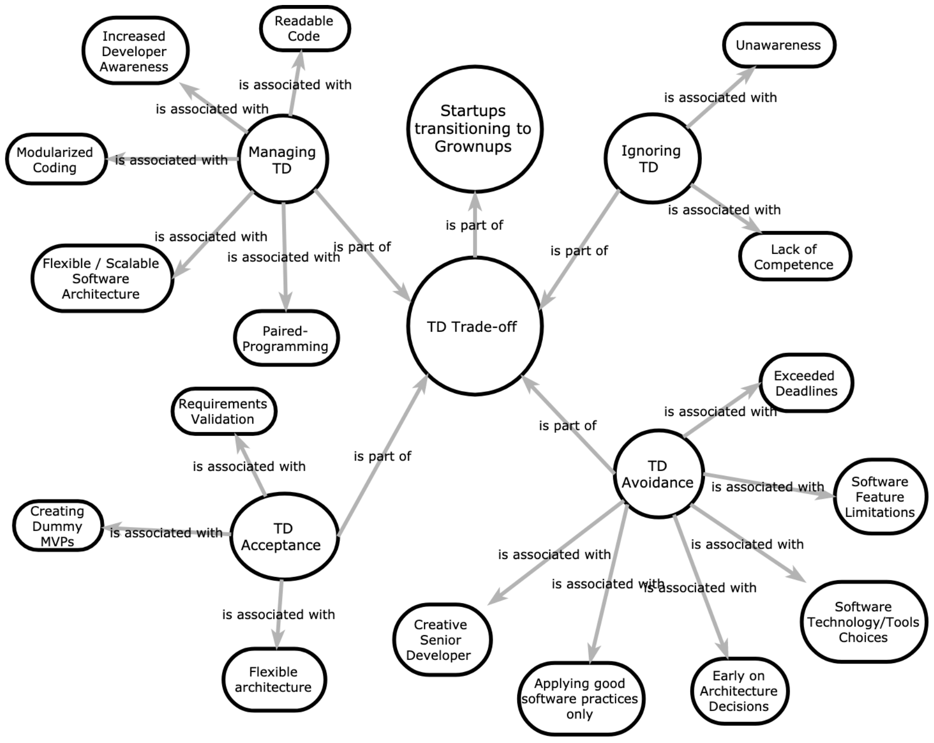

Increased Developer Awareness
Readable Code
Modularized Coding
Managing TD
Flexible / Scalable Software Architecture
Paired-Programming
Startups transitioning to Grownups
TD Trade-off
Ignoring TD
Unawareness
Lack of Competence
Requirements Validation
Creating Dummy MVPs
TD Acceptance
Flexible architecture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

Fig. 2. Themes summary.

TD (Section 4.4), and Ignoring TD (Section 4.5), each helping to answer our RQ in the following subsections.

4.1 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 we need to make in achieving deadlines. Our team is highly deadline driven."*. Yet another case, number 3, told the following when asked if they wanted to avoid TD: *"No, we need to move fast. We need to try things and them. So we decided to make a trade-offs."*. We obtained similar answers from the rest of the cases. 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.

4.2 Managing TD

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 paired programming approach to software development. Whenever a developer has deliberately introduced TD to certain modules, he would raise a flagship to make other team members aware of the situation; more precisely : *"We sort the story cards and set aside team time to simply address technical debt. Everybody on the team tends to know in the project what is the state of the code and where are the challenges or things to be solved. So if somebody says hey we're going to write a story card its going to change this part of the code. Somebody might raise their hand and say hey we really need to improve that code."*

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. The same case also emphasizes the possibility to mitigate TD whenever having backend software architectures, which can be easily modified and scaled. Furthermore, we found a recurring pattern, readable code, from all cases, and that was made part of managing TD. Readable code allows the developers to quickly unfold whenever TD is introduced to the software system, without requiring extensive documentation. This approach was considered of immediate relevance from all our cases under investigation.

4.3 Avoiding TD

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. Although, it was commonly accepted from all cases that a trade-off has to be accepted and that it was hard to totally circumvent scenarios introducing TD. Highly creative senior level developers are perceived as a particular case that can elegantly fulfill the requirements while introducing marginal levels of TD. Case 1 mentions that it is enough to tell them why you are building it and them help you figure out what to build.

4.4 Accepting TD

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."*, although with drawbacks discussed in the earlier Section 4.3. However, in Case 4 the CEO stated the following in very early startup phases: *"We took lots of shortcuts. Mmhmm. So it was all about shortcuts to get to the first prototype sooner. Okay. So at that point we didn't care about robust engineering."*. Continuing statement at a later stage of their startup development: *"But now that we are getting bigger, we are trying to use good software techniques and like to make sure that everything is robust. We are trying to make all the code follow all the different guidelines. But, we still agree we need to take shortcuts and make trade-offs from time to time."*

4.5 Ignoring TD

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. Nevertheless, ignoring TD might still be part of trade-offs in startups when still in PoC phase and software is not in production stage. case 2 elegantly reported this scenario while stating: *And we were pretty convinced that we could just do this. And we were told over and over again by manufacturing just no, you can't. It's really hard. When we finally realized, after taking several shortcuts, oh, crap, this is really hard. We can't do this or it's going to take two years if done appropriately. The failed prototype allowed us to learn internally about factors towards achieving our goal.* Thus, despite not being fully aware about the TD, even when they became aware, ignoring it at this phase proved to be efficient and looking for competence elsewhere, joint venture, seemed to be the successful option.

Key findings:

1. All startups transitioning to grownups stages accept that TD trade-offs are crucial and widely accepted, although they have different approaches to cope with TD. (**Section 4.1**)
2. 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. (**Section 4.2**)
3. 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. However, it is commonly accepted by most of the cases that avoiding TD in sacrificing field validation can bring more harm than benefit to the startups. (**Section 4.3**)
4. 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. (**Section 4.4**)
5. Ignoring TD is primarily affected by lack of awareness and lack of competence. (**Section 4.5**)

5 Discussions

5.1 Early-stage technical debts vs. Grownup technical debts

In this section, we compare our findings with existing knowledge on TD in software startups. Although the study focuses on a particular niche context, startups transitioning to grownups, we find our results to have unfolded some important unnoted differences from previous sources. Thus, we can provide startups unique recommendations. Nevertheless, limitations of the study exist and are also mentioned in this section.

Many of the previous studies have focused on covering and addressing several startup life cycle phases by unfolding the TD challenges and benefits [3]. 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 [3].

We believe that TD while transitioning to grown up company has a different perception compared to TD while at very early stage. Despite the risk here being bigger the companies use their experience to make more deliberate decisions in avoiding, managing, accepting or even ignoring TD.

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.

5.2 Threats to validity

The study focuses on highlighting perceptions about TD in software startups transitioning to grownup stages. Although, we have a limited number of participants, the qualitative nature of the study permitted us to obtain legitimate results that focus on deeply understanding the perceptions rather than evaluating them on a superficial level.

As often reported in qualitative research [22] main threats to validity related to :

– **External Validity.** Related to the sample size and limited context under consideration. We mitigated this validity while choosing software startups before, during, and after transitioning to grownup stages. This phase distribution fits more to our study than choosing a broader range of startup segments or geolocations. Same applies to the roles of the interviewees who are purposely selected to be co-founders of the startups, with major stakeholder concern in the product/service and startup success. Although this doesn't generalize the results obtained as of yet, we plan in the future to recruit further samples from different areas and use data triangulation (follow up questionnaires) to improve our understanding of the data.
– **Internal Validity.** Internal threats to validity in qualitative studies are related to data extraction and analysis. We tried to mitigate this threat in our case by carefully coding and categorizing the transcriptions and gradually shrinking to the most significant data.
– **Construct Validity.** Construct validity in our cases is related to the previous knowledge about TD. Nevertheless, the maturity level of the startups

proved that they were all very well acquainted with the concept, and this threat to validity was almost non-existing.

6 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 also provides a set of recommendations, becoming an initial roadmap that can support startup decisions if they need to transition into TGs.

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.

References

[1] P Abrahamsson et al. “Software Startups - A Research Agenda”. In: *e-Informatica Softw. Eng. J* 10.1 (2016), pp. 1–28.

[2] Vebjørn Berg et al. “Software Startup Engineering: A Systematic Mapping Study”. In: *Journal of Systems and Software* (2018).

[3] Terese Besker et al. “Embracing Technical Debt, from a Startup Company Perspective”. In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2018, pp. 415–425.

[4] Virginia Braun and Victoria Clarke. “Using thematic analysis in psychology”. In: *Qualitative research in psychology* 3.2 (2006), pp. 77–101.

[5] Nanette Brown et al. “Managing technical debt in software-reliant systems”. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM. 2010, pp. 47–52.

[6] Mark V Cannice. “Confidence among Silicon Valley Venture Capitalists Q3 2017–Q4 2018: Trends, Insights, and Tells”. In: *The Journal of Private Equity* 22.3 (2019), pp. 18–24.

[7] Mark Crowne. “Why software product startups fail and what to do about it. Evolution of software product development in startup companies”. In: *IEEE International Engineering Management Conference*. Vol. 1. IEEE. 2002, pp. 338–343.

[8] W Cunningham. “The WyCash portfolio management system, Experience Report”. In: *Proceedings on Object-oriented programming systems, languages, and applications (OOPSLA’92)* (1992).

[9] Nicolas Devos, Dimitri Durieux, and Christophe Ponsard. “Managing technical debt in IT start-ups–an industrial survey”. In: *International Conference on Software and System Engineering and their Applications (ICSSEA).* 2013.

[10] Max Garkavtsev, Nataliya Lamonova, and Alexander Gostev. “Chosing a Programming Language for a New Project from a Code Quality Perspective”. In: *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP).* IEEE. 2018, pp. 75–78.

[11] Carmine Giardino, Xiaofeng Wang, and Pekka Abrahamsson. “Why early-stage software startups fail: a behavioral framework”. In: *International Conference of Software Business.* Springer. 2014, pp. 27–41.

[12] Carmine Giardino et al. “Software Development in Startup Companies: The Greenfield Startup Model”. In: *IEEE Transactions on Software Engineering* 42.6 (2016), pp. 585–604.

[13] Catarina Gralha et al. “The evolution of requirements practices in software startups”. In: *Proceedings of the 40th International Conference on Software Engineering.* ACM. 2018, pp. 823–833.

[14] Yuepu Guo et al. “Tracking technical debt. An exploratory case study”. In: *2011 27th IEEE international conference on software maintenance (ICSM).* IEEE. 2011, pp. 528–531.

[15] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. “Technical debt: From metaphor to theory and practice”. In: *IEEE software* 29.6 (2012), pp. 18–21.

[16] Zengyang Li, Paris Avgeriou, and Peng Liang. “A systematic mapping study on technical debt and its management”. In: *Journal of Systems and Software* 101 (2015), pp. 193–220.

[17] Anh Nguyen-Duc, Xiaofeng Wang, and Pekka Abrahamsson. “What Influences the Speed of Prototyping? An Empirical Investigation of Twenty Software Startups”. In: *Agile Processes in Software Engineering and Extreme Programming.* Ed. by Hubert Baumeister, Horst Lichter, and Matthias Riebisch. Lecture Notes in Business Information Processing. Springer International Publishing, 2017, pp. 20–36.

[18] *Nvivo Homepage.* `https://www.qsrinternational.com/nvivo/home`. Last accessed 16 Aug 2019.

[19] AL Penenberg. *Eric Lies is a lean startup machine.* 2011.

[20] Eric Ries. *The lean startup: how today’s entrepreneurs use continuous innovation to create radically successful businesses.* 2011.

[21] N Robehmed. *What is a startup? Forbes.* 2013.

[22] Per Runeson and Martin Höst. “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical software engineering* 14.2 (2009), p. 131.

[23] Carolyn Seaman and Yuepu Guo. “Measuring and monitoring technical debt”. In: *Advances in Computers.* Vol. 82. Elsevier, 2011, pp. 25–46.

[24] Edith Tom, AybüKe Aurum, and Richard Vidgen. “An exploration of technical debt”. In: *Journal of Systems and Software* 86.6 (2013), pp. 1498–1516.