

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

The Negative Implications of Technical Debt on Software Startups: What they are and when they surface

Abdullah Aldaej

Department of Management Information Systems
Imam Abdulrahman Bin Faisal University
Dammam, Saudi Arabia
aaaldaej@iau.edu.sa

Carolyn Seaman

Department of Information Systems
University of Maryland Baltimore County
Baltimore, United States
cseaman@umbc.edu

ABSTRACT

Technical Debt (TD) refers to suboptimal technical solutions for expediting software development in the short term, but entails extra work in future. Although TD provides opportunities for startups (e.g., to cope with limited resources, faster time to market), it can have some negative impacts throughout the startup evolution. While previous studies have investigated the effects of TD, there is little understanding of the conditions under which TD effects can surface. In this paper, we conducted a multiple case study to explore the effects of TD incurred in the early startup stage, and when these effects are felt later in the evolution of the startup. First, we interviewed 17 participants (6 CTO/CEO founders, and 11 developers) from five software startups. In addition, we analyzed public documents related to the five software startups. Based on the results, we identified six TD effects, along with the time (i.e., the startup evolution stage) and the conditions under which they are felt by software startup teams. Our results can help startup teams to understand not just the TD effects, but also when they are likely to surface and their relationship to customer dissatisfaction.

CCS CONCEPTS

• Software and its engineering → Software design tradeoffs;
Software product lines; Maintaining software

KEYWORDS

Software startups, Technical debt, Technical debt effects

ACM Reference format:

Abdullah Aldaej and Carolyn Seaman. 2022. The Negative Implications of Technical Debt on Software Startups: What they are and when they begin to surface. In *Proceedings of the 5th International Workshop on Software-intensive Business (IWSiB 2022)*. Pittsburgh, PA, USA, May 18, 2022. DOI: <https://doi.org/10.1145/3524614.3528629>

© 2022 Association for Computing Machinery. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

IWSiB'22, May 18, 2022, Pittsburgh, PA, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-9302-7/22/05...\$15.00

<https://doi.org/10.1145/3524614.3528629>

1 Introduction

Software startups are new/immature businesses that focus on the development of an innovative software product or service [1]. Startups often have unique contextual factors in comparison to other organizational context. First, a software product is usually developed with high uncertainty about the market requirements [2]. Since there are few or no real users or operational history, the software requirements are usually not well understood in startups. More time needs to be spent on user/customer discovery than building an optimal solution. In addition, startups often have very limited resources which lead them to rely on a few novice developers [3]. Given this unique context, startups often pursue a suboptimal design and implementation with less emphasis on long-term product sustainability. This condition describes the concept of Technical Debt (TD).

TD is defined as the gap between the suboptimal and optimal technical solutions, more specifically in internal software quality [4]. This gap can be larger in software startups due to their unique contextual factors. Although accumulating TD has short term advantages for startups (e.g., faster time to market), it can have negative impacts in the long term. Previous studies investigated TD effects in diverse organizational context. Some of the TD effects are delaying the delivery of new releases [5],[6], increasing maintenance effort [5],[6],[7],[8], and deteriorating the team productivity and morale [9],[10]. Sometimes these negative impacts are exacerbated and lead to customer dissatisfaction [5],[11].

The effects of TD have been studied intensively in previous research. However, such research has been mostly focusing on mature companies that have been in the market for a long time. There is lack of study about TD effects in startups. Since the effects of TD usually appear in the long term, it is worth investigating what TD effects, and when they surface in startups. In this paper, we report emerging results of our research project about TD decision making in software startups. This emerging results are related to the effects of TD incurred in the early startup stage and when they are revealed during startup evolution. We analyzed the data within the lens of this scope (i.e., TD effects), applying qualitative analysis, primarily thematic analysis. Based on the results of this analysis, we identified six themes related to TD effects in software startups. Also, we pinpointed the time and conditions when these effects

surface (i.e. become evident to the development team) throughout the startup evolution. Our results provide insights that help startup teams to understand the conditions under which TD effects can be felt, and hence support TD management and decision making in startups.

The remainder of this paper is structured as follows: Section 2 presents related work about TD effects and TD in software startups. In section 3, we explain our research methodology. Section 4 reports our results, which are discussed in section 5. Section 6 concludes the paper.

2 Related Work

2.1 TD Effects

Previous studies revealed some negative consequences of TD that can affect the business in the long term. The main effect is on internal software quality, primarily maintainability and evolvability. Accumulating TD increases code complexity and fragility, which can increase maintenance cost [6], [7]. Although TD directly impacts internal software quality, it has indirect impacts on other aspects of the business. Once the software has low maintainability, the team performance and productivity can be reduced [3], [12]. This means that extra time will be needed to perform major changes in the software. Once the team productivity is reduced, it can negatively impact customer satisfaction [11], [13]. In addition, poor internal quality such as code complexity increases the amount of rework and amplifies the risk of introducing extra defects and ripple effects [3], [11]. The exacerbation of the TD impacts might in some cases affect the developers' morale [10], the team retention [12], and the overall company performance [14].

2.2 TD and Software Startups

A software startup is a young company that focuses on the development and management of a single software product or service [15]. In comparison to other organizational contexts, startups have a unique evolution model [16], [17], that begins when it is founded and ends when it becomes a mature company. During this timeline, startups move through three different stages: Inception, Stabilization, and Growth. The first stage (Inception) extends from the startup's founding time until releasing the initial version of the product (i.e., MVP) to market. Then, startups enter the second stage (Stabilization) and remain there until they find the product-market fit. After that, the third stage (Growth) begins where startups scale up their businesses until they achieve the required market share or net-profit (i.e., maturity level). In this paper, we call this timeframe a "startup timeline" to represent the startup duration more precisely, instead of using the term young organizations.

Giardino et al. [1] investigated the software development strategy employed by startups using the grounded theory approach. Based on the empirical findings from 13 startup cases, the authors created the "Greenfield Startup Model" that characterized software development in startups. Speeding up development was found to be strongly associated with the accumulation of TD. Faster delivery of

software to meet market demand is vital in startups. However, the study only investigated software development in early-stage startups, primarily stage-1 (Inception). Our paper more specifically explores TD throughout a wide spectrum of the startup evolution from founding to maturity.

Klotins et al. [18] explored the contextual aspects in software startups that amplify the effect of TD. They surveyed 86 startup cases in North America, South America, Europe, and Asia. The study found that team size and experience are the main contextual factors that contributed to the accumulation of TD in startups, which is the same finding in [3]. The effect of team size and experience become more severe as the startup evolves. The lack of experience can be acceptable in the early stage as its impact is low, but in the later stages (i.e., growth) its impact becomes more severe. Also, the presence of TD becomes a barrier that might prevent the startups from delivering new features faster, and gaining new clients. In particular, architecture and code debt are the TD types that most significantly contribute to this impact. This study focuses on contextual factors related to TD whereas our paper investigates TD effects from deliberate TD in startups (i.e., the effects of TD that are accumulated intentionally in the early stage).

Gralha et al. [19] identified some conditions that trigger when TD should be managed and remediated. TD is accepted in the early startup stage. Hiring more developers and expanded new feature implementation plans are a trigger for the need to track and report TD. Under these conditions, startup teams need to report and track which and how features are affected by TD. When the client's retention rate decreases and more negative customer feedback is received, it is a trigger for action to control or remediate TD. However, the study explores TD from the perspective of requirement engineering, and discusses triggers at a high abstract level. In this paper, we identify specific finer-grain conditions that trigger TD effects.

Cico et al. [20] interviewed 13 CEOs and CTOs from seven startups to investigate how TD perception changes over the startup evolution stages. The study found that startups in the early stage accept and ignore TD due to different factors such as requirement validation and lack of developer competence. In the late stage, startups change their perception of TD by managing and avoiding TD. This can be via adopting good programming practices, proper tools, and proper software architecture. This study is similar to ours in the sense that it explores a phenomenon retrospectively throughout the startup timeline. However, it concentrates on the startups' perception of TD, whereas our study investigates the effects of TD that intentionally accumulated in the early startup stage. A summary of the related work, in comparison to our paper, is presented in Table 1.

Table 1. Comparative Summary of Related Work

Literature domain	The focus of literature	The focus of our paper
TD effects in general (Section 2.1)	<ul style="list-style-type: none"> Not specific to startup organizations 	<ul style="list-style-type: none"> Specific to startup organizations

TD and software startups (Section 2.2)	<ul style="list-style-type: none"> • Factors influencing TD accumulation • Startups' perception of TD • High-abstract TD effects that are only associated with the growth stage 	<ul style="list-style-type: none"> • TD effects organized based on conditions and startup stages • TD effects from deliberate TD
----------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------

3 Research Method

This multiple case study aims to understand the effects of TD on software startups, and when TD effects can be revealed throughout the startup evolution. To address this goal, we derive the following research questions (RQs):

RQ1: What are the effects of TD on software startups?

RQ2: When do TD effects become visible for software startup teams?

The targeted cases had to satisfy the following criteria:

- The case must be a software startup, i.e. a new software organization that develops and owns (partially or fully) a software product or service.
- The case must be a minimum of one and maximum of ten years old since the founding date, with preference for cases closer to ten years old.
- There must be at least two interviewees available for the case, including a founder.

We relied on convenience sampling by contacting founders of software startups within our network and contact list. In addition, we used snowball sampling by asking each founder to forward our invitation message to their team members, or to allow us to reach them through LinkedIn, as well as to founders of other software startups. At the end, we were able to select five cases, presented in Table 2, in the order in which they were selected. All the cases are currently operating companies that own and develop a web or mobile-based software product. Three of our cases are late-stage startups. Startup-2 was at the end of its growth stage (i.e., almost reached maturity) whereas Startup-1 and Startup-5 have been at the growth stage for two and four years respectively. We have two cases that are early-stage startups (Startup-3 and Startup-4). These startups are at the beginning of stage 2, in which their software product was released and had been available in the market for several months.

Table 2. Description of Cases

Case	Startup stage	Product	Application domain	Location (Continent)
Startup-1 (founded in 2012)	Late-stage startup	Web app	Enterprise Resource Planning (ERP)	Asia

Startup-2 (founded in 2010)	Late-stage startup	Mobile and web app	Education	North America
Startup-3 (founded in 2019)	Early-stage startup	Mobile app	Transportation	Europe
Startup-4 (founded in 2019)	Early-stage startup	Mobile and web app	Medical	Asia
Startup-5 (founded in 2012)	Late-stage startup	Mobile and web app	Retail	Asia

We interviewed a total of 17 participants from five cases. Table 3 presents our study participants. All the founder participants had software engineering knowledge or experience (i.e., technical founder). For example, the founders of startups 1, 2, and 3 had a degree in computer science or other related fields before beginning their startups. In addition, the founders of startups 4 and 5 had prior experience as software developers before beginning their startups. For the employee participants, we counted their experience in their role (not limited to the time of their involvement in the startup). Most of the employee participants were software developers, and a few of them were systems analysts.

Table 3. Study Participants

ID	Role	Gender	Experience in this role	Case
P1	Founder & CTO	Male	8 years	Startup-1
P2	Senior developer	Male	7 years	Startup-1
P3	Developer	Female	3 years	Startup-1
P4	System analyst	Male	6 years	Startup-1
P5	System analyst	Male	1 year	Startup-1
P6	Founder & CTO	Male	10 years	Startup-2
P7	Developer	Male	1 year	Startup-2
P8	Developer	Male	2 years	Startup-2
P9	Android developer	Female	7 years	Startup-2
P10	Founder & CTO	Male	2 years	Startup-3
P11	Developer	Male	1 year	Startup-3
P12	Founder & CEO	Male	6 years prior experience: 3 years as a software developer	Startup-4

P13	Founder & CTO	Male	2 years <i>prior experience:</i> 8 years as a software developer	Startup-4
P14	Developer	Male	2 years	Startup-4
P15	Founder & CTO	Male	8 years <i>prior experience:</i> 2 years as a software developer	Startup-5
P16	IOS developer	Male	5 years	Startup-5
P17	Developer	Female	6 years	Startup-5

3.1 Data Collection

Our primary data collection method was semi-structured interviews. We interviewed 17 participants (6 technical founders and 11 technical employees), starting with a founder in each of the five cases, between April and October 2020. Our initial recruitment was via a direct message to software startup founders known to us. This invitation message contained some brief information about the study and a link to an online form where the participant could register. We started with 11 candidate startup cases and interviewed a founder in each of these cases initially. Through snowballing, we successfully recruited other interviewees for 5 out of the 11 candidate cases, resulting in the final set of cases presented in Table 2. The remaining 6 founder interviews were excluded since they did not satisfy our study design criteria (i.e., triangulation of at least 2 participants per case). For each of the five selected cases, we were able to interview at least one technical employee. All of the technical employees had been involved in their cases for at least six months.

All the interviews were conducted online (via Skype or Zoom) and lasted between 40 and 70 minutes. We applied two interview instruments, one for founders and one for employees. Instrument I was used for founders. Since founders usually oversee all the evolution stages of their startups, this instrument includes some opening questions about the startup's initial story and main objectives. It also identifies the startup evolution stages by asking the founders about key milestone events. For example, we asked when the initial product (i.e., MVP) was released to market, when the product-market fit was obtained, etc. On the other hand, instrument II was used for technical employees who often joined the startup after it was founded. This instrument includes some opening questions to identify the timeframe in which the employee was involved in the startup. This helped us to limit the discussion to the startup stages in which they participated. All the instruments

are available in the supplemental material ¹.

Both of the interview instruments are structured in three parts. The first part includes the opening questions to collect contextual information about the startup, as described above, and the participant. In the second part of the interview, we presented the three startup evolution stages [16],[17]. Then, we asked participants how these stages applied to their startups. For the founder participants, we captured the startup timeline by determining the approximate beginning and ending dates for each stage. For the employee participants, we first mapped their time of involvement to the startup timeline that was previously captured by the startup's founder. In the third part of the interview instruments, we investigated decisions about TD in chronological order (per startup stage). First, we presented the technical debt concept to participants based on the TD definition in the Dagstuhl report [21]. In addition, we used the TD landscape [22] [7] to illustrate the scope of TD. We presented this information to 1) make sure that all the study participants had a common understanding of the TD concept, and 2) mitigate the risk of collecting invalid TD decisions.

We conducted three pilot interviews for the purpose of validating the interview instrument. The participants for the pilot interview were two researchers who also had software engineering work experience, and one founder and CEO of a software startup. As a result of the pilot, we made some modifications to the instrument to improve the interview flow and the understandability of the questions.

In addition to the interviews, we used some public documents (i.e., secondary data) related to each case such as the company website, the company profiles in social media and in startup community reports. In some cases, we used the information available in the mobile app platform such as app reviews and ratings. We used this secondary data to triangulate some of the interview data. For example, some TD effects from the interview data (e.g., customer satisfaction, bug rate) were triangulated with secondary data from app review/rating, and release note.

3.2 Data Analysis

Prior to the analysis, all data collected from the interviews were recorded and transcribed. We read the interview transcripts several times. Then, we created separate sub-files for each decision about TD that were discussed in the interview transcripts. Each sub-file contains all data related to a TD decision. For this paper, we only use the data related to the effects of TD incurred in the early startup stage, and the context at the time TD effects are felt. So we only focus on TD accumulated in the early startup stages. Then, we mapped TD effects to their associated TD.

We applied thematic analysis to generate themes about TD effects, and the context at the time when the TD effects were felt. Since the TD effects usually appeared after the TD was accumulated, we collected additional context (e.g., product conditions and constraints) at the time when the effects were felt.

¹ <https://drive.google.com/file/d/1xHlCqWU96zlo5IXPyh4MNURvxyspPLo/view?usp=sharing>

Also, we mapped the context of each TD accumulation decision with its related TD effects.

4 Results

Our analysis revealed six commonly observed categories of TD effects in startups. It also indicates when the TD effects are felt throughout the startup stages, and the conditions that trigger them. In the remainder of this section, we briefly explain each of the six TD effects we found. Then, we present how some of the TD effects can lead to customer dissatisfaction and churn in software startups.

4.1 TD Effects on Software Startups

4.1.1 Increased Post-release Bugs. After releasing the initial software version to market, startups usually observed many bugs in the production environment (post-release bugs). *“The features looks wow from the outside but when they were deployed to the production, problems surface. Startup-2, P-6”*. This TD effect is felt in the second stage when the usage traffic increased. We found that test debt was the main contributor to this effect. *“If you don’t have test covering most of the code, you are not going to solve issues. Startup-1, P-2”*.

4.1.2 Extra Time to Perform Maintenance Activities. TD causes startups to spend more time than usual to complete a maintenance task (e.g., bug fixing, change impact analysis). Many types of TD contribute to this effect at different points in the startup timeline. However, we found that this TD effect is mostly observed in stage 3 when the code size increases, primarily from design and architecture debt. But it could also be felt early in the second stage when either of two conditions occur: usage traffic increased or the team is less familiar with the existing code/tool. Test debt can also contribute to this effect, in some cases. We found that test debt led to less confidence when performing some changes during the second stage, especially when user traffic increased. When this condition is reached, it forces the startup teams to allocate more time for analyzing the change impact before releasing the change. As explained by one participant, *“It brings lack of confidence. Every time that we change something, for example in the API, you have to think a lot about if this change breaks something in the mobile application. Startup-3, P-10”*. In addition, using unfamiliar code/tool can lead to this TD effect early in stage 2. As illustrated by some participants, *“we reuse an open source software and try to make some customizations to it. The topic was very complicated, and take a lot of learning curve. Startup-1, P-2”*. Another case adopted a new technology that was not familiar to anyone on the team. *“We used some new technologies only because it is new. After that, the maintenance become difficult. Startup-5, P-15”*.

4.1.3 Inability to Perform Enhancements. TD could prevent startups from performing some required enhancements (e.g., UI enhancement, updating the development framework) in an optimal way. This restriction stems from TD that intertwined between architecture and infrastructure debt (e.g., full-stack development framework). This TD effect is felt early in the second stage when startups need to build custom UIs. Sometimes, the adopted (sub-

optimal) development infrastructure restricts customized changes in the UIs. For example, the WebNotes framework limited Startup-1’s ability to add new UIs based on customer feedback. *“Our UIs are inherited from the WebNotes framework. We tried to add new UIs, but that was very difficult tasks. Startup-1, P-1”*. In addition, the inability to enhance the UI can be observed in a mobile app that is developed using web or hybrid architecture. For example, Startup-4 used a hybrid framework (Ionic) while knowing its limitations for enhancing mobile UIs. *“Currently we don’t have control over the UIs of our mobile app because it is a responsive view of the website. Startup-4, P-12”*. Another enhancement issue that startups could encounter is related to upgrading the development framework. We found that this impact occurred in the third stage after the code size significantly increased. After making large changes in the code while skipping a series of framework versions, startups found it more challenging to upgrade the development framework.

4.1.4 Evolution Restrictions. TD could prevent startups from evolving their software in an optimal way or considerably delaying the implementation of new features. Similar to the enhancement ability restriction, this effect is more associated with architecture and infrastructure debt. This type of restriction mostly appeared in the third stage when the infrastructure or architecture was outdated. For example, Startup-1 could not develop an advanced feature within the outdated WebNotes framework. *“We decided to stop implementing new features. Startup-1, P-1”*. The evolution restriction could sometimes occur early in the startup timeline. However, this kind of restriction was often caused by unexpected limitations of an existing infrastructure or architecture. For example, Startup-3 observed this effect in the first stage. *“We realized that the MySQL wasn’t so good at handling geographical data. Startup-3, P-10”*. Also, Startup-4 recognized this effect in the second stage. *“We used customer.io, and tried to do automation of the user journey for both side the clinics and patients. At the end we discovered a gap that we cannot do this automation for two sides. Startup-4, P-12”*.

4.1.5 Infrastructure Cost. All the startup cases in this study built their architecture/infrastructure based on a third-party cloud provider. Although the cost of cloud-based infrastructure usually correlates with data size, accumulating architecture/infrastructure debt may further increase the infrastructure cost in the cloud-based context. We found that the effect of infrastructure cost is visible when startups scale up in the third stage while using outdated architecture/infrastructure. For example, the single-tenancy architecture amplified the infrastructure cost in Startup-1. *“The development environment is built as single tenancy that as you know it is a VM per customer which is expensive in comparison to multi tenancy environment that allows you to share the host infrastructure. Startup-1, P-1”*. Also, the outdated version of the CakePHP framework forced Startup-2 to incur extra infrastructure cost by adding extra infrastructure resources that would not have been needed if the latest version of CakePHP were used. *“We talk about performance issues which means cost on the infrastructure. For example, to be able to serve 10K concurrent users to 20K or*

whatever. Using the CakePHP version 2, I need more infrastructure to serve them. Startup-2, P-6”.

4.1.6 Difficult-to-write Test Code. We found that writing tests for previously written code becomes extremely difficult in the third stage. This effect stems from the accumulation of both test and design debt, often when the code size increased significantly. For example, Startup-1 could not write tests for the old code during the third stage, mainly due to the design and test debt. “Since some period until now, we suffer to improve the quality and write automation tests since we have old code and very low test coverage. Startup-1, P-1”. Our analysis revealed an association between design and test debt during the TD repayment process. The difficulty to write test originates from complex or fragile code design. So it was always necessary to refactor the code in order to write a test for it. For example, improving test was collocated with code refactoring in the third stage in both Startup-2 and Startup-5.

Figure 1 summarizes the six TD effects over the startup evolution stages. It shows when and under what conditions each effect was felt by the startup team, and the TD types that contributed to each effect.

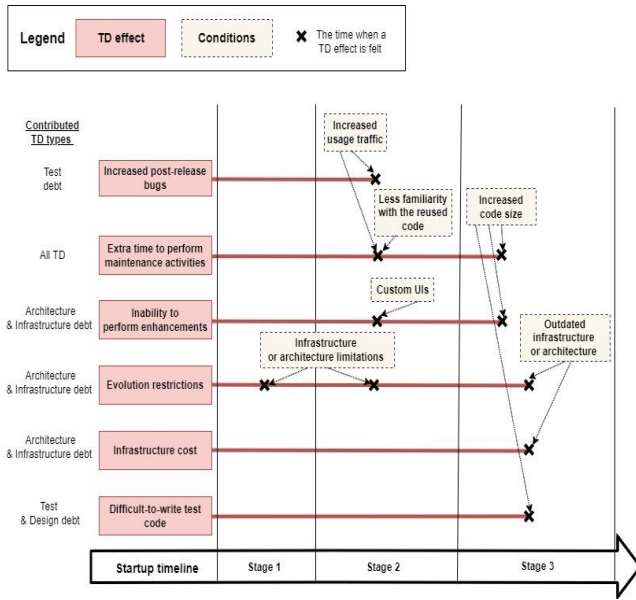


Figure 1: TD Effects throughout Startup Evolution Stages

The TD effect *Increased post-release bugs* is felt in the second stage when the usage traffic is increased. Also, the TD effect *Extra time to perform maintenance* is revealed in the second stage when either the usage traffic is increased or the team is less familiar with the reused code. *Evolution restriction* can surface at any point of time. In the early stage (stage 1 and 2), it is primarily triggered by unexpected limitations at the infrastructure/architecture level. But in the late stage (stage 3), it is mostly caused by outdated infrastructure/architecture. Other TD effects such as *Infrastructure cost* and *Difficult-to-write test code* are triggered by outdated infrastructure/architecture and increased code size respectively.

4.2 TD Effects and Customer Dissatisfaction

We found two scenarios where TD effects led to customer dissatisfaction and churn in software startups. The first scenario appeared in the early startup stages when the two TD effects (*More post-release bugs* and *Extra time to perform maintenance activities*) occur together. So compromising the maintainability early (e.g., reusing unfamiliar code while skipping test) would put startups at a high risk of customer dissatisfaction in the beginning of stage 2. The second scenario could lead to customer dissatisfaction in the late startup stage. This scenario could be observed if startups bypass some observed evolution restrictions by implementing many workarounds. The additional TD introduced because of the evolution restrictions might deteriorate the system performance in terms of non-functional requirements (e.g., poor latency.). If so, startups can be vulnerable to customer dissatisfaction. Figure 2 illustrates how some TD effects contribute to customer dissatisfaction and churn in software startups.

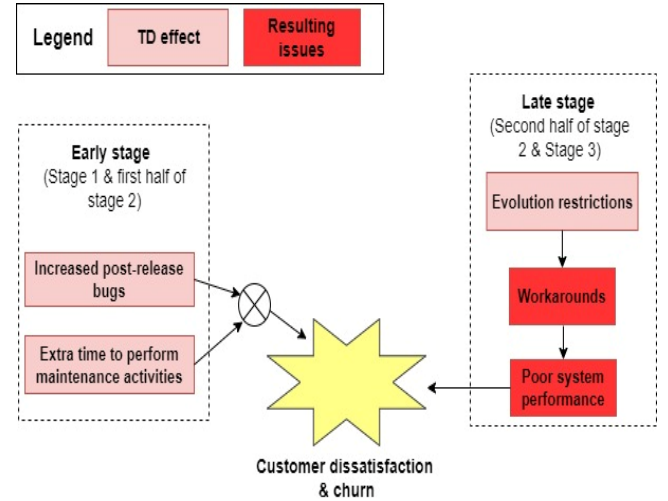


Figure 2: Relationship between TD Effects and Customer Dissatisfaction and Churn

5 Discussion

Most of our identified TD effects were found by previous studies [3],[6],[18]. First, the TD effects *Extra time to perform maintenance* and *Evolution restriction* are found in the TD literature [5], [9], and software startup literature [3], [18]. This is expected since TD is directly associated with the impact on maintainability and evolvability. Another TD effect is *more bugs*, which was also found in [5]. However, we found that early-stage startups perceived this impact differently than other organization contexts. In the early-stage startups, this usually refers to bugs in the production environment. For this reason, we used the label *Post-release bugs* to reflect the startup perception for this effect. Finally, the TD effect *difficult-to-write test* is also found in [5]. However, it was explained there as part of the evolution restriction (in the context of development issues). This was true when testing was part

of the development process, which is often not the case in most of the startup timeline. In startup context, the effect of test difficulty was observed once startup teams decide to add test code for the previously developed features.

The main contribution of our results, in comparison to related work, is that our results characterize (rather than just identify) TD effects. First, we identified some contexts at the time when TD effects are observed. Second, we conceptualized the relationship between TD effects and customer dissatisfaction in software startups.

5.1 Implications

Our results have implications for both practitioners and researchers. Practitioners can benefit from our conceptualization of TD effects, their triggers, and their relationship with customer dissatisfaction to better support their TD decision making. Also, we provide more insights for researchers about TD effects by identifying some conditions at the effect time. Future researchers who study TD effects are encouraged to collect some information about the context at the time when the effect is observed. For example, the code size or some other product metrics at the effect time, in comparison to the time when the debt was accumulated. When investigating the effect of TD, it is important to know that there are two different time dimensions: 1) the time when the TD is introduced, and 2) the time when its effect is observed. Collecting contextual information at the two times helps to identify some metrics that correlate with the TD effects, and hence, improve the actionability of the findings.

5.2 Limitations

There are some limitations and threats to the validity of our results. First, we investigate TD effects retrospectively by considering TD incurred intentionally in the beginning of software startups. This approach can affect the participants' ability to recall past experience. So it is possible to miss some accumulated TD. This threat was mitigated by 1) focusing on major accumulated TD that have observed impacts, 2) beginning data collection for each case by interviewing the case founder who has deeper insight of all major decisions since the founding time, and 3) applying data triangulation by interviewing other members in the development team who joined the startup in different timeframes.

In addition, our identification of the time and conditions associated with TD effects can be subjective. The time associated with TD effects is based on participants' approximation of when they experienced them. Also, the identified product conditions depends on what the participants' said while explaining the TD consequences. We mitigated this threat by triangulating data from public documents such as app reviews/rating and the app's social media account. However, this can only be used for certain issues such as customer satisfaction and bug rate. Future studies can mitigate this threat by triangulating the interview data with some product metrics (e.g., software quality metrics) at the time of TD effects.

Finally, our results of TD effects are limited to the TD items that are deliberately introduced from decisions discussed in the interview. There might be other "unintentional" TD items (that the participants were not aware of) that influenced the TD effects found in the study. Also, it is possible that other factors (beyond TD and software engineering) contribute to the TD effects such as management or organization issues or team conflict, etc.

6 Conclusions

Software startups usually suffer from many constraints such as limited resources and product uncertainty, which lead them to ignore TD until they face its negative consequences. We conducted a multiple case study of five startup cases, to understand what and when TD effects observed in software startups. First, we interviewed 17 participants from the five startup cases to explore the effects of TD accumulated in the early startup stage. Then, we mapped these effects to the startup contexts at the time when they are felt by the startup teams. Our results present the common TD effects in software startups, and the time (within the startup evolution stages) and product conditions that trigger these effects (Figure 1). Also, we illustrate two scenarios where TD effects can lead to customer dissatisfaction and churn in software startups. In future work, we plan to conduct an intensive single-case study to examine our findings in more depth.

ACKNOWLEDGMENTS

We would like to thank all participations who devoted their valuable time for this study. Selecting this diverse set of startup cases could not have happened without the support of researchers such as Xiaofeng Wang, Anh Nguyen, Pekka Abrahamsson, and Mohammed Alshayeeb.

REFERENCES

- [1] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software Development in Startup Companies: The Greenfield Startup Model," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 585–604, Jun. 2016, doi: 10.1109/TSE.2015.2509970.
- [2] E. Klotins, M. Unterkalmsteiner, P. Chatzipetrou, T. Gorschek, R. Prikladnicki, N. Tripathi, and L. Pompermaier, "A progression model of software engineering goals, challenges, and practices in start-ups," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019, doi: 10.1109/TSE.2019.2900213.
- [3] T. Besker, A. Martini, R. E. Lokuge, K. Blincoe, and J. Bosch, "Embracing Technical Debt, from a Startup Company Perspective," in *International Conference on Software Maintenance and Evolution*, Madrid, Spain, 2018, p. 12.
- [4] M. Ciolkowski, V. Lenarduzzi, and A. Martini, "10 Years of Technical Debt Research and Practice: Past, Present, and Future," *IEEE Software*, vol. 38, no. 06, pp. 24–29, Nov. 2021, doi: 10.1109/MS.2021.3105625.
- [5] N. Rios, R. O. Spinola, M. Mendonça, and C. Seaman, "The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil," *Empir Software Eng*, Jun. 2020, doi: 10.1007/s10664-020-09832-9.
- [6] R. Ramač, V. Mandić, N. Taušan, N. Rios, S. Freire, B. Pérez, C. Castellanos, D. Correia, A. Pacheco, G. Lopez, C. Izurieta, C. Seaman, and R. Spinola, "Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry," *Journal of Systems and Software*, p. 111114, Oct. 2021, doi: 10.1016/j.jss.2021.111114.

- [7] P. Kruchten, R. Nord, and I. Ozkaya, *Managing Technical debt - Reducing friction in software development*. Pearson Education, 2019.
- [8] J. M. Conejero *et al.*, "Early evaluation of technical debt impact on maintainability," *Journal of Systems and Software*, vol. 142, pp. 92–114, Aug. 2018, doi: 10.1016/j.jss.2018.04.035.
- [9] T. Besker, A. Martini, and J. Bosch, "Software Developer Productivity Loss Due to Technical Debt - A replication and extension study examining developers' development work," *Journal of Systems and Software*, Jun. 2019, doi: 10.1016/j.jss.2019.06.004.
- [10] T. Besker, H. Ghanbari, A. Martini, and J. Bosch, "The Influence of Technical Debt on Software Developer Morale," *Journal of Systems and Software*, p. 110586, Apr. 2020, doi: 10.1016/j.jss.2020.110586.
- [11] J. Yli-Huumo, A. Maglyas, and K. Smolander, "The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company," in *Product-Focused Software Process Improvement*, 2014, pp. 93–107.
- [12] N. Rios, R. O. Spínola, M. G. de Mendonça Neto, and C. Seaman, "Supporting Analysis of Technical Debt Causes and Effects with Cross-company Probabilistic Cause-effect Diagrams," in *Proceedings of the Second International Conference on Technical Debt*, Piscataway, NJ, USA, 2019, pp. 3–12, doi: 10.1109/TechDebt.2019.00009.
- [13] E. Lim, N. Taksande, and C. Seaman, "A Balancing Act: What Software Practitioners Have to Say about Technical Debt," *IEEE Software*, vol. 29, no. 6, pp. 22–27, Nov. 2012, doi: 10.1109/MS.2012.130.
- [14] R. Banker, Y. Liang, and N. Ramasubbu, "Technical Debt and Firm Performance," *Management Science*, July. 2020, doi: 10.1287/mnsc.2019.3542.
- [15] V. Berg, J. Birkeland, A. Nguyen-Duc, I. O. Pappas, and L. Jaccheri, "Software startup engineering: A systematic mapping study," *Journal of Systems and Software*, vol. 144, pp. 255–274, Oct. 2018, doi: 10.1016/j.jss.2018.06.043.
- [16] M. 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*, Aug. 2002, vol. 1, pp. 338–343 vol.1. doi: 10.1109/IEMC.2002.1038454.
- [17] E. Reis, *The lean startup*. New York: Crown Business, 2011.
- [18] E. Klotins, M. Unterkalmsteiner, P. Chatzipetrou, T. Gorschek, R. Prikladnicki, N. Tripathi, and L. Pompermaier, "Exploration of Technical Debt in Start-ups," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, May 2018, pp. 75–84.
- [19] C. Gralha, D. Damian, A. I. T. Wasserman, M. Goulão, and J. Araújo, "The Evolution of Requirements Practices in Software Startups," in *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg, Sweden, 2018, pp. 823–833. doi: 10.1145/3180155.3180158.
- [20] O. Cico, R. Souza, L. Jaccheri, A. Nguyen Duc, and I. Machado, "Startups Transitioning from Early to Growth Phase - A Pilot Study of Technical Debt Perception," in *Software Business*, Cham, 2021, pp. 102–117. doi: 10.1007/978-3-030-67292-8_8.
- [21] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, pp. 110–138, 2016, doi: 10.4230/DagRep.6.4.110.
- [22] P. Kruchten, Robert. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software*, vol. 29, no. 6, pp. 18–21, Nov. 2012, doi: 10.1109/MS.2012.167.