Universiteit Leiden
The Netherlands

Computer Science & Economics

Truck factor analysis for business continuity planning in SMEs

Kevin Kraayeveld

Supervisors:
Joost Visser & Paul van Leeuwen

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/06/2023

Abstract

Business continuity planning (BCP) is the process of proactively minimizing the impact of possible disasters on an enterprise. One such disaster is the loss of important knowledge about the enterprise. The loss of knowledge is especially prevalent in small to medium-sized enterprises (SMEs) in software development. Despite the need for a well-developed BCP, SMEs often struggle with this. In this thesis, we propose a design that should ease the development of the BCP regarding the threats of knowledge loss. To achieve this, we use a tool called the truck factor (TF). The truck factor is a way of quantifying the continuity threats of a software company in terms of knowledge loss. The truck factor is defined as the number of developers that have to be hit by a truck for the project's continuity to be in danger. To estimate the truck factor of a software project we use an algorithm which is currently considered to be the most accurate and accessible. For a more complete BCP design, we extend the current best BCP approach for SMEs with the truck factor tool specifically for software development enterprises. To validate the design we evaluated it with a Dutch software development company. We asked the opinion of our design people with different roles in this company, ranging from owner to software developer. From this evaluation process, we found that people who are higher up in the company usually have a few major problems with the design, while employees more involved with the development are generally positive. The problems with the design that they mentioned are mostly properties and missing functionalities of the truck factor algorithm itself. To address the alleged shortcomings, a handful of changes should be implemented on the truck factor to make it more practically valuable. With the changes, the tool should be more useful for people in managerial roles as well as improve the accuracy of the truck factor results given by the algorithm.

Contents

1 Introduction

1.1 Problem formulation

Business continuity planning (BCP) is an indispensable tool to increase the longevity of a firm. Business continuity management is the process of taking proactive measures to guard information systems against disasters [JN+22]. For companies in the IT business, this is obviously of incredible importance, since information systems are the core of their business. Without the information systems, their product or service does not exist. Most approaches to developing business continuity plans don't apply well to small to medium-sized enterprises (SMEs). We will explain the reasons for this in section 2. To define what an SME is, we consult the European legal definition. The European Union legally defines an SME as an organization with less than 250 employees [Uni03]. Because SMEs often struggle with business continuity and its immense importance for software development organizations, we aim to provide a BCP design specifically tailored for SMEs in the software development business.

1.2 Research approach

To address the problem defined in section 1.1, we will use the current best practice BCP process for SMEs. On top of that, we will make it specific to software development enterprises by extending the current best practice with a tool called the Truck Factor (TF). The Truck Factor is a measure used to quantify the concentration of knowledge of a group of developers in a software development project [FVF17]. Therefore, can be used to mitigate continuity risks from knowledge loss. The Truck Factor is usually calculated by using data from Git repositories because Git stores data on who wrote and edited each line of code. This data can be used to determine the amount of knowledge each developer has per file. The Truck Factor is also known as the Bus Factor or the Lottery Number. The Truck Factor is defined as the number of developers that have to be hit by a truck (or abruptly leave the project in any other way) for the project's continuity to be endangered [FVF17]. As a rule of thumb, the higher the Truck Factor is, the lower the continuity risks. A high Truck Factor means knowledge is highly distributed over the software development team. If one person were to leave the project, it shouldn't cause a business disruption in that case. Currently, there is little to no data on the practical applicability of the truck factor. Because the truck factor will be a large part of our BCP design, that is the most important aspect of this thesis. That gives us the following research question:

RQ: How valuable is the truck factor tool for small to medium-sized software development enterprises as a part of business continuity management?

1.3 Thesis overview

This bachelor thesis is done as a part of the Leiden Institute of Advanced Computer Science and supervised by prof.dr.ir. Joost Visser. For this thesis, we will first set up the theoretical framework this research is built upon in section 2. In this section, we investigate research done in the BCP process for SMEs to find the current best practice. We will also look at comparative studies to find the current best algorithm for the Truck Factor and a threshold for the truck factor.

This thesis is done as an internship for a small Dutch software development company to improve the business continuity of their firm. Section 3 explains how this thesis will follow the structure of design science research and how we plan to answer the research question. In section 4 we will formulate the design including the truck factor. The evaluation results of the design will be presented in section 5. We will interpret these results in section 6. Finally, we will answer the research question in section 7. We draw these conclusions from the results we have gathered and discussed.

2 Theoretical Framework

2.1 Business continuity planning for SMEs

Making a BCP is essentially investing time and resources into something you wish does not happen. Apart from that, assuming you will have to use it some time, it is not clear when. Therefore, investing in a BCP is unattractive, since you don't know when you'll yield the returns on your investment. However, there is a clear need to develop a BCP, especially for SMEs [Nat15]. Despite this need, research and frameworks for business continuity planning are typically aimed at large corporations [KS18], [Her10].

There are several reasons why SMEs can't simply adopt the frameworks developed for larger corporations. First is the fact that SMEs simply have to work with fewer available resources [STB11]. SMEs usually lack resources in finance, technology, and human personnel. [Vos98]. The second reason these frameworks do not work well for SMEs is the fact that SMEs often have a less bureaucratic work environment [BR93] than big corporations. Even though this is seen as an advantage by some [Vos98], it still causes problems when implementing a BCP framework developed for a more bureaucratic, larger firm. Other differences between SMEs and larger firms are; SMEs can make decisions faster, which makes them more flexible, and internal communication is easier and faster within SMEs due to the less bureaucratic work environment [Vos98]. Faster decisions and communication cause SMEs to learn more quickly and adapt routines and strategies quicker as well [Vos98]. Additionally, the organization's size usually makes the environment less formal [SSSG+10]. This calls for a strategy that develops a BCP which is informal, not overly bureaucratic, and can be developed with fewer resources. Järvelainen et al. have developed such a strategy [JN+22]. They call it the Thrifty BC management approach.

The Thrifty approach has proposed the following ten meta-requirements (MR) to make a successful BCP for SMEs [JN+22]:

1. Facilitate embeddedness. Make business continuity management a part of the organizational process.
2. Pay attention to context. As mentioned before, the nature and circumstances of the company are important factors to take into account when developing a BCP.
3. Maintain accuracy. As the company evolves, the BCP has to be continually updated to accurately address current continuity risks.
4. Develop gradually. Develop a plan for the most threatening risks first and develop the lesser threats later to spread the use of resources over time.
5. Minimize documentation. Enforcing excessive documentation creates a hierarchical environment, which is the opposite of the nature of SMEs [Her19], [VS11]. That is not to say that documentation is not useful, it just should not be used excessively.
6. Enable self-assessment and development. Do not over-complicate the BCP. Organizations should be able to develop and implement the BCP without outside counsel.
7. Develop the social and technical jointly. The technical aspect is *what* should be done to manage BC, and the social aspect is *who* should manage it.
8. Facilitate collective participatory development. Involve people with different roles in the company.

9. Revere substance expertise. Make BC decisions based on expertise, not on hierarchy.
10. Attend proactively. Risk mitigating measures should be developed before incidents happen.

These meta-requirements aim to reduce the expense of resources and suit the informal environment of SMEs better. The meta-requirement most relevant to this thesis is MR6 (enable self-assessment and development) because it relates to the use of the Truck Factor. The Truck Factor is a self-assessment tool. It is used to assess the knowledge concentration of a software project and the risks caused by that concentration.

To implement the 10 meta-requirements the approach has designed a process consisting of 3 phases [JN+22]:

1. Map-it
2. Design-it
3. Continue-it

The map-it phase aims to identify the most critical business process(es) and other context of the particular business such as the most probable risks. In the design-it phase, the BC measures are defined. In the continue-it phase, firms decide how to act and monitor the BC measures. To achieve the goals of each phase, they are broken down into modules. All three phases combined have 16 modules as shown in figure 1.

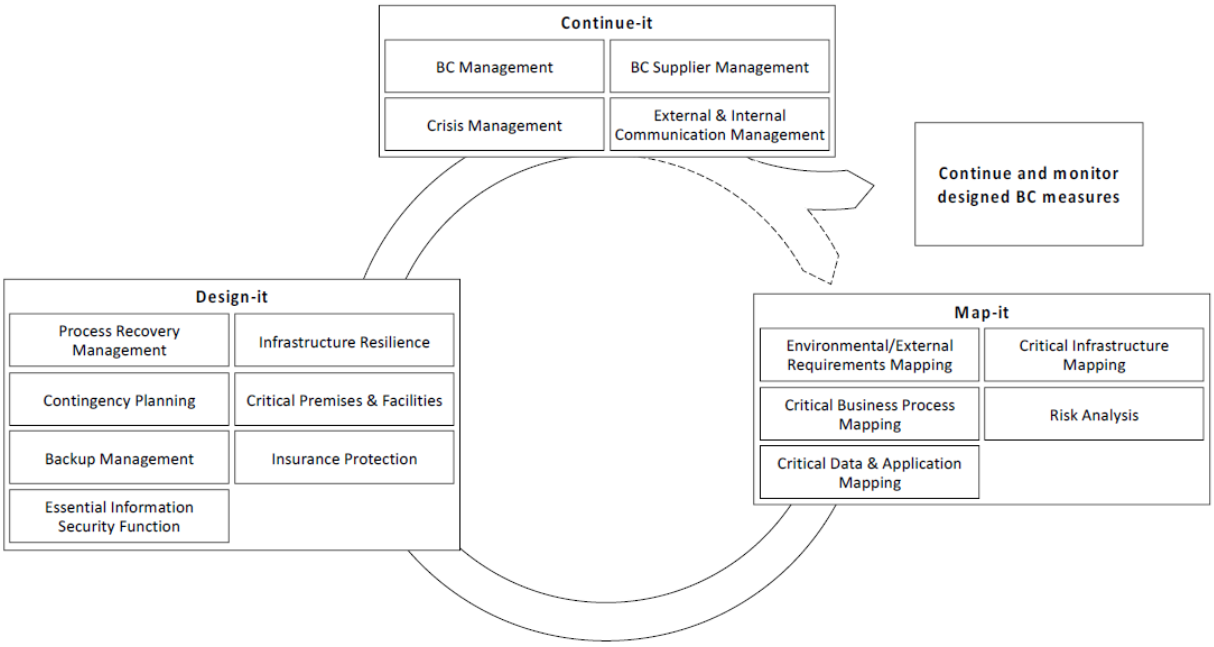

Figure 1: The SME development design proposed by Järvelainen et al.

2.2 Truck Factor

Since 2010 a few algorithms have been proposed to estimate the truck factor of a software project. The first is an algorithm proposed by Zazworka et al. [ZSK+10]. A comparative study to find the

best algorithm has been done by Ferreira et al. [FVF17]. This study compared three algorithms. The first of these algorithms is proposed in a part of a study by Avelino et al. [APHV16]. The second algorithm has been proposed by Rigby et al. [RZDM16]. The third algorithm evaluated in this paper has been proposed by Cosentino et al. [CIC15]. The research has chosen not to evaluate the algorithm proposed by Zazworka et al. because it has major scaling problems. Research by Ricca et al. has concluded that the Zazworka algorithm does not scale for projects past 30 developers [RMT11]. Because we established the definition of an SME as an enterprise that could have up to 249 employees, we also won't consider the algorithm proposed by Zazworka et al [ZSK+10].

In the research done by Ferreira et al. [FVF17] they test the three algorithms on two accuracy measures. The first one is the TF value accuracy, that is to say, how close the value given by the algorithm is to the actual value. The second accuracy measure is how accurate the identification of the TF developers is. A TF developer is a core developer the project cannot lose, therefore part of the Truck Factor. They test this because the algorithm could give the correct TF value, but the wrong people could be in the TF set. The research concluded that the algorithm proposed by Avelino et al. [APHV16] is the most accurate in both TF accuracy and TF developer accuracy.

The comparative study was done in 2016 [FVF17]. Since 2016 one new TF estimation algorithm has been proposed by Jabrayilzade et al. [JETK22]. For simplicity, we will call this the JAB algorithm. In their research, they claim to have built a slightly more accurate algorithm than the AVL algorithm [APHV16] which was established as the state-of-the-art by the comparative study [FVF17]. On the 13 projects they tested the algorithm on, they established a mean absolute error of 5.46, compared to a mean absolute error of 5.80 for the AVL algorithm [JETK22]. However, they do not provide all the data which led them to this result. In addition, to estimate the Truck Factor, the JAB algorithm uses unavailable and unfeasible data for most software projects. They argue knowledge of code comes from meetings as well as writing code. Therefore they also use the number of minutes a software engineer spends in meetings to the variables to estimate the Truck Factor for that project [JETK22]. That makes this algorithm impractical because, without data of all previous meetings between software engineers, you cannot use this algorithm. For these reasons, we have decided not to use the JAB algorithm in this thesis but to use the AVL algorithm instead.

We have established that AVL is the best TF algorithm, but that does not mean it is without shortcomings. The AVL algorithm is perfectly accurate if the TF $= 1$ (for 20 systems). For higher TF values, the accuracy does drop quite drastically. For $2 \leq$ TF ≤ 5 (13 systems), the accuracy Ferreira et al. [FVF17] found was 30%, and for TF ≥ 6, they found an accuracy of 50%, although the size of this group was only 2. Therefore the results most likely aren't an absolute accuracy number for TF ≥ 6. However, these results indicate that the algorithm is imperfect if TF ≥ 2. We do need to keep this in mind for our thesis.

The AVL algorithm [APHV16] computes the TF by simulating the departure of important developers of Git repositories. To do this, you need to know which developers are essential. The AVL algorithm does that by calculating the degree of authorship (DOA) a developer has for every file [APHV16]. When a file f is created by developer d_1, a $DOA(d_1, f)$ value gets initialized. Additional commits to that file by the same developer increase the $DOA(d_1, f)$ value. Commits by another developer (d_2) decrease the $DOA(d_1, f)$ value and increase $DOA(d_2, f)$ value. After running over

all the commits on a file, the $DOA(d, f)$ values for all its authors have been calculated. These $DOA(d, f)$ values then get normalized as a value between 0 and 1. The developer with the highest DOA has their value normalized to 1. To be considered an author of file f, a developer d needs to have a minimum normalized $DOA(d, f)$ of 0.75. The algorithm simulates the removal of top authors until 50% of the files become abandoned. A file is abandoned when all authors of the file have been removed by the algorithm. Finally, the Truck Factor equals the number of authors that have to be removed before that threshold is met. In the comparative study, 50% has been confirmed to be the threshold with the best accuracy [FVF17]. The source code of the AVL Algorithm is available on GitHub[1].

2.3 Code Ownership

The AVL algorithm [APHV16] holds a pretty heavy assumption: code authorship is equal to code ownership. To validate the usage of the AVL Truck Factor algorithm in the context of knowledge distribution, it is important to confirm that code authorship does correlate with code knowledge. A study by Fritz et al. has concluded that frequency and recency of interaction with a piece of code do indicate the level of knowledge a developer has on that code [FMH07]. In this study, 19 developers were given a questionnaire about the functionality of certain pieces of code. The results pointed out that developers had memorized more of the code they wrote themselves [FMH07]. All interviewed developers also said most correct answers they gave were about code they wrote themselves [FMH07]. This research proves that code authorship does mean knowledge about the code and therefore validates the use of the AVL algorithm [APHV16].

2.4 Truck Factor threshold

Because it is unclear how high the truck factor for a repository should be, it is critical to have a threshold to compare it to. To determine the threshold, we will look at available literature regarding this subject. Although there has been research regarding the TF threshold, no viable formula or model is available. The only available study on this topic is by Torchiano et al., in which they evaluate a threshold proposed by Siddhi Govindaray [TRM11]. The Govindaray threshold states "*Small teams of under 10 people usually target a TF of 4-5 for most parts of the system (that is around 40-50% of the team). Larger teams will probably target a TF of around 8 (which would probably be around 20-25% of the team)*"[2]. In the evaluative study, they found this threshold was too high and often even impossible to achieve in real situations [TRM11]. That means there is no truck factor threshold defined in the literature.

[1] https://github.com/aserg-ufmg/Truck-Factor

[2] https://siddhi.blogspot.com/2005/06/truck-factor.html

3 Methodology

To answer the research question we will create a design to implement the truck factor into the business continuity management process of a software development SME and evaluate its practical usability. The development of the design will follow the structure of design science research [PTG+20]. This structure consists of 6 parts: problem identification, objectives of a solution, design, and development, demonstration, evaluation, and communication. The problem identification and solution have been presented in section 1. The design will consist of running an algorithm to acquire the truck factor results of all repositories of an organization. Because we have established in section 2 the AVL algorithm [APHV16] is the best, that is the algorithm we will be using in our design. Due to a lack of a truck factor threshold definition in the literature, we will define one ourselves in the design. The execution of the truck factor algorithm and comparing it to the threshold is part of the map-it phase of the thrifty BC management approach [JN+22]. When a truck factor is lower than the threshold, that means the risk is too high. That means some form of measures have to be taken to mitigate that risk. Therefore we will also propose a list of measures in the design. Those measures are part of the design-it phase of the thrifty BC management approach [JN+22]. Finally, for the design, we will suggest a period after which the design should be repeated after it has been implemented. This is part of the continue-it phase of the thrifty BC management approach [JN+22]. To evaluate the design, we will demonstrate it to five people from the software development company we partnered with for this thesis. We will demonstrate it to the owner of the company, the manager, the functional administrator, the technical administrator, and a software developer. We purposely ask people with different roles in the company to facilitate collective participatory development of the continuity plan, which is meta-requirement eight [JN+22]. After demonstrating it, we will interview the five people to which we demonstrated the design. The interview questions refer to the design, so will formulate the questions after the design, in section 5. The interpretation of the opinions given the in the interviews will allow us to conclude whether the design is practically valuable for this particular company. The key indicator of whether it is practically valuable is if the company wants to implement it. Whether the design will be implemented or not is up to the management team, so their opinions will weigh a bit higher than others. The entire design process is shown in figure 2.

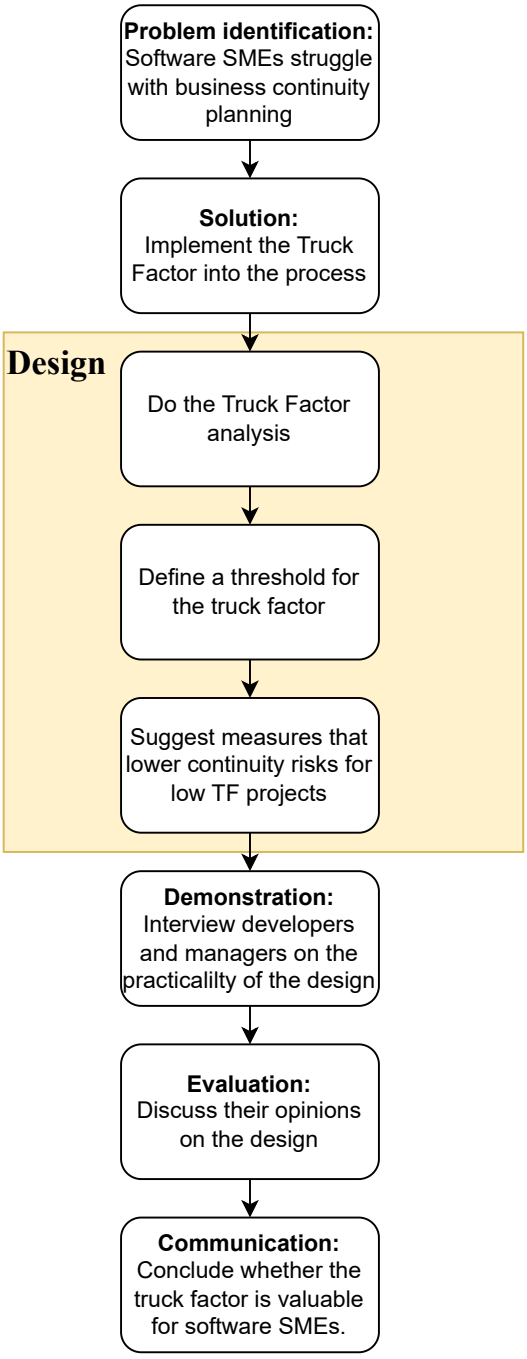

Figure 2: Design approach

4 Design

4.1 Running the algorithm

First of all, the truck factor has to be calculated for all repositories of an organization. As we have established in section 2, the algorithm proposed by Avelino et al. [APHV16] is the most accurate and accessible at the moment. To run the algorithm, follow the instructions given on the GitHub page. After the truck factor results have been gathered, the following question might arise: "Is this truck factor high enough?" To give more meaning to the truck factor the next step is to compare it to a threshold.

4.2 Truck Factor threshold

Because there is no concrete literature available to determine the truck factor threshold, we are obligated to propose one ourselves. Instead of providing a single threshold, we suggest a range of truck factor scores a project could fall under. We distinguish risk scores for truck factors of 0, 1, 2, 3 and 4 or more. We go up to four or more, because a truck factor of four already means four people have to leave the company simultaneously for the continuity to be endangered. Because we estimate the chances of that happening as very small, no distinction has to be made with a truck factor higher than four. Having a truck factor score of 0 means the project's continuity is currently at great risk. All of its important developers have left the project.

We define the threshold as a risk score, rather than taking the truck factor as the only input for the truck factor. We also take the business value of a repository into account to calculate the risk score. We define high business value as a business component that would cause the company to go out of business within a month if it does not function properly or at all. A malfunctioning business component of medium business value would also cause the company to go out of business but in a bigger time frame. This means a business component of medium business value is still crucial for a company, but wouldn't cause immediate problems. Finally, a business component with low business value would not cause a company to go out of business, but would only cause some inconvenience or annoyance for customers or other users of the software. Categorizing projects based on business value is useful because SMEs probably do not have the resources to improve the business continuity of all projects. This is in line with one of the meta-requirements of the thrifty approach [JN+22], which is to pay attention to context and focus on the most critical business processes. The business value of a project is something each firm has to determine on its own. When the business value is determined and the truck factor has been calculated, a risk score can also be determined. The risk score of a project based on the truck factor and business value can be seen in table 1.

Note that we have not taken into account the size of the organization in the TF threshold. That means that a project with only 2 developers, for example, will always be at medium risk at best. This clear definition of risk scores distinction shows where measures are most urgent. This allows the gradual implementation of the measures, which is in line with the fourth meta-requirement, which states business continuity management should be implemented gradually [JN+22]. The risk scores given in table 1 indicate to which extent risk-minimizing measures should be taken for a certain software project. The definitions of the risk scores can be seen in table 2.

Risk Score		Truck Factor				
		0	1	2	3	4+
Business value	1	3	3	2	1	1
	2	3	3	2	2	1
	3	3	3	2	2	1

Table 1: Risk Score

Risk score	Measures
1	No measures needed
2	Consider taking measures
3	Measures are absolutely needed

Table 2: Measures based on risk score

4.3 Measures to decrease TF risks

Now that we have defined when measures should be taken, we will elaborate on what these measures should look like. To decrease the risks of having a low truck factor there are two options. The first option is to simply try to increase the truck factor. The second option is to develop alternative measures that would decrease the likelihood of business disruption when all TF developers were to leave the project.

To increase the truck factor, an organization has a few options. The most simple option is to hire more developers. However, more often than not, this is the least feasible option for SMEs. Because if an SME is concerned about its business continuity and truck factor it probably already tried to hire new developers, but didn't have the resources or new developers weren't available. There is also an option to increase the truck factor with the existing development team. For this option, the organization could try to shift employees from projects with a very high truck factor (four or more) to projects with a low truck factor. That way, the organization can smooth out its risk without hiring new people. However, this approach is often impractical, because shifting employees is not always possible. There might not be a single repository with a truck factor of 4 or more. Or the different projects require different skill sets, which makes shifting employees between them impractical.

The alternative to increasing the truck factor is to implement other measures in the development process. In a survey developers of large GitHub projects were asked which practices would be most effective in decreasing the risk of business discontinuation in a truck factor event [APHV16]. The 5 practices that the developers mentioned the most were (out of 111 answers):

1. Documentation (36)
2. Have an active community (15)
3. Automatic tests (10)
4. Code readability (10)
5. Code comments (7)

As can be seen, documentation is by far the most common answer to the questionnaire. Although the importance of documentation is widely known, a lot of projects still lack documentation [VvGMW16]. Many developers see documentation as a burden rather than an important part of software development [VvGMW16]. That means developers likely won't write documentation if they're not encouraged to do so. We established earlier that SMEs most often aren't very bureaucratic. So simply forcing developers to write documentation does not fit and therefore likely won't work. Because a forceful approach doesn't work for SMEs, writing documentation has to be embedded in the culture of an organization. If, for example, the organization's culture is very result-oriented, documentation is likely to be overlooked. Therefore, an organization might have to change its culture to motivate developers to write documentation.

The most straightforward way to achieve this culture change is to make documentation one of the steps of the development process. This approach is in line with the first meta-requirement, which is to facilitate embeddedness [JN+22] When it is part of the process, managers can expect a slightly slower pace of development, and developers know when they have to do it. Note that when managers still expect projects to be developed just as fast as before documentation was being written, it is not going to work. But what sort of documentation should be written? For a software development company, the most important thing to maintain continuity is keeping existing software operational. Therefore debugging code and maintaining existing software are most important. Therefore documentation should support these activities. For the task of debugging code, developers find code comments the most useful [ANLV+20]. For all maintenance tasks a user manual of the code is one of the most useful documentation artifacts, according to a survey done on developers [ANLV+20].

The second most common answer to the questionnaire is to have an active community. In the context of SMEs having an active community means having developers communicate with each other frequently. This communication could be encouraged by letting developers give frequent updates of their projects to the other developers and doing code reviews. Besides improving code quality, code reviews can make developers more familiar with their colleague's code [BR15].

The third most common answer is to run automatic tests on the code. Automatic tests help find bugs faster when a change has been made. This is especially useful to a person who does not have much knowledge about the code they're modifying because it's clear whether something has been broken or not. However, much like writing documentation, writing tests costs time and might feel like a burden to software developers. Here the culture of an organization is also important and goes hand in hand with the "documentation writing culture". To not overburden developers with software testing we have to prioritize which tests are most important for continuity in case of a Truck Factor event. As we have mentioned earlier, debugging and code maintenance are the most important tasks to maintain continuity in case of a truck factor event. Unit tests and integration tests are most important for these tasks. Writing unit tests has proven to significantly reduce the need for debugging because unit tests help isolate a bug and detect it earlier in the development process [Ola03]. While unit tests can detect bugs at the micro-level, there still might be issues with the integration of different components of the software at a macro-level. Unit tests are not able to detect these bugs, but integration tests can. Therefore, using both sorts of testing goes hand in hand and provides a more complete approach to testing a software system.

The last two most common answers are closely correlated. Code comments are often just a way to make code that is difficult to read more readable. Apart from making code more readable, code comments should state assumptions developers make in the code because what is evident to one developer, might not be to another [ANLV+20]. To make code more readable, code reviews might be useful here as well. As well as improving the familiarity of other developers' code we established earlier, code reviews also significantly improve the readability of code that has been reviewed as opposed to code that has not been reviewed [BR15].

4.4 Timeline

Since maintaining business continuity is a process, it should be repeated after a certain amount of time or at a certain event. Repeating the process is important because the truck factor can change over time. One of the most prominent developers leaving the company has an impact on the truck factors that the developer was active in. An event like that should be a trigger for the process to start over. That means calculating the truck factors for the different repositories and applying measures where needed. The truck factor changing could also be more subtle. When one developer has been gradually spending more time on another project than before, that developer could no longer be a part of the initial project's truck factor. For that reason, we suggest repeating the process every 6 months.

5 Design evaluation

To validate the design, we have asked the opinions of 5 people at the software development company we have partnered up with. Their opinions are supposed to give an insight into whether the design is practically useful or not. To go in depth on the opinions of validation group, the validation will be in the form of an interview. To get a wide range of viewpoints from the validation group we will interview people from different roles in the company. We will interview the owner of the company, the manager, the functional administrator, the technical administrator and a software developer.

In the interview, the following questions will be asked:

- Do you think the use of the truck factor is valuable for the company?
- Do you think the suggested measures actually work to minimize continuity risks?
- Should there be more measures than currently suggested?
- Are the suggested measures feasible?
- Are there other concerns with regards to knowledge loss which this design does not cover?
- How accurate do you think the truck factor algorithm is, based on a few examples?
- Do you like the idea of making the minimizing of continuity risks into a process?
- Do you think this design could create more awareness about continuity among the employees?
- Do you think 6 months is a reasonable timeline?

Some questions are more relevant for certain people and less relevant for others, based on their role in the company. That means the interview will be semi-structured. Follow up questions will be based on where each person is most knowledgeable and where they hold the strongest opinions.

Due to working with time limits, some questions of the interview were not answered by everybody. However, we were sure to ask questions most relevant to that person's role in the company. The most important thing is to know whether they value the use of the truck factor or not. More technical questions about the measures, for example, are not as relevant for everyone, because not everyone has the technical knowledge to answer these questions comprehensively. The design validation is divided into five sections, as a way to structure the answers given in the interviews. The first one is truck factor usage. This section shows the opinions on the truck factor to decrease continuity risks for their company. The second section, algorithm accuracy, indicates whether the interviewees' expectations on the truck factor align with the results. The third presents the opinions on the measures proposed in the design meant to decrease continuity risks. The fourth section elaborates on a few new measures suggested by the interviewees. The last section shows their opinions on the process as a whole.

5.1 Truck factor usability

Three out of five people said the truck factor is valuable for the company. The owner and the functional administrator stated that the truck factor is not detailed enough. He argues most repositories can be broken into multiple functional components. The truck factor does not differentiate between these components. The owner indicated that knowing the truck factor for each component of a repository would be more valuable due to the increased understanding of identified risks. According to both people, seeing where the continuity risks are for the company at a repository level is

pretty straightforward. They think the use of the truck factor is unnecessary. To identify continuity risks, the owner suggested that it would be better to list each technical component manually and assign people with knowledge to each component. Because of the lack of detail, the functional administrator thinks the truck factor in its current form is only valuable for larger companies. Larger companies have more employees, which makes it harder to identify knowledge loss risks, he reasoned. Only then could the truck factor in its current state be valuable. In his opinion, an SME would only benefit from the truck factor if it provided more details.

Opposingly, in the technical administrator's opinion, the truck factor does not need to be more detailed than it is. He agrees with the functional administrator that you don't exactly know which part of a repository is at risk from the truck factor alone. However, he still values the use of the truck factor very much. As he explained, even though a truck factor developer may not know everything about a repository, he does know enough about the structure and functionality of the repository to figure it out relatively quickly. Therefore, knowing the truck factor of each component is not needed to make informed decisions, the truck factor in its current form is sufficient.

5.2 Algorithm accuracy

Although most people's expectations of the truck factor results matched the actual results of the algorithm, the software developer did not fully agree with the algorithm's estimations. According to the developer, the result for one specific repository was wrong. Even though he has the most commits in that repository, he was not part of the truck factor. He also provided a reason as to why that may be the case. He stated that the company's code base was migrated to Git after years of development. Therefore, he thinks the algorithm does not work well in this particular case.

5.3 Proposed measures

Apart from a few additions and remarks, everyone was satisfied with the proposed measures. Everyone agreed documentation would help decrease continuity risks. The functional administrator thought the documentation measure needed guidelines to define what should be documented. He also said the documentation should include as many images and other ways to visualize the technical architecture as possible. He stated that visuals are easier to understand than large amounts of text. He mentioned that making visual representations of a software system might require more thought from the author of the documentation and therefore more time, but documentation that is hard to understand is useless. Therefore it is worth the extra effort.

Everyone was also a proponent of creating a more active community to decrease continuity risks. Everyone thought of doing code reviews as an adequate way to achieve a more active community. The technical administrator suggested extending the collaboration in software development even further. Rather than only reviewing each other's code, he suggested discussing how developers will realize a functionality with at least one other developer before writing the code.

Everyone also thought tests were a good measure but with a few conditions. The tests should be automatic and well-named. Automatic tests ensure you get notified when previously written code breaks because of a poorly written addition in the code. The unit tests should be well-named so

the bug is easy to find. If the name of the unit test is vague, it is unclear what piece of code has just broken, which slows down the debugging process. That defeats part of the purpose of having unit tests. Everyone thought unit tests and integration tests were sufficient and would decrease continuity risks. The functional administrator even valued these tests above documentation in their ability to maintain continuity risks.

Although everyone agreed code readability and code comments are essential, not everyone agreed on our suggestion on when to use code comments. We mentioned in the design, code comments are mostly unnecessary when code is easily readable. The technical administrator argued that code comments' purpose is to explain *why* certain pieces of code are being executed, rather than explaining what it does. He argued this explanation helped refactor code in the future, since the purpose of the code is clear, future refactoring is not bound to the structure of the old code. He agreed code should be self-evident by well-named variables and short functions, but he valued code comments more than the initial design did.

5.4 Additional measures

In the interview, we asked the interviewees whether they had any other concerns regarding continuity this design does not cover and whether they had any additional measures they deemed useful. In response to that, they suggested two new measures. Instead of permanently hiring a new employee, a company can hire a third-party developer for a repository with a low truck factor who can help in times of crisis. Note that this does not increase the truck factor of this particular repository, but could help solve problems more quickly. However, this measure still requires other measures, such as documentation and code readability, otherwise, this third-party developer cannot understand the code either and will not be of any help.

The technical administrator also suggested a new measure. This measure is closely related to the code readability measure. Namely, it is about code structure. He argues a good code structure could also help the debugging process. Proper code structure makes the program more logical as a whole. This logic should save time trying to find malfunctioning code and therefore reduce the risk of prolonged downtime of the software system.

5.5 Process

Almost everyone agreed with the idea of continually repeating the process proposed in the design to manage continuity risks. Only the functional administrator came back to his standpoint given earlier. He believed the design did not fit an SME, meaning there is no reason to repeat the process. He argued that it was easy enough to see when continuity is at risk at a small company. According to him, regularly reviewing the current state of risks at the company is unnecessary, because the risks are clear. Everyone else thought repeating the process was a good idea, although there were a few different thoughts about the proposed timeline. The technical administrator said six months is a decent amount of time to repeat the process, although he predicted the proposed measures would take longer than that to carry out. Nevertheless, he still likes having a regular time interval for repeating the process and mentioned it should never be longer than a year. The software developer had the idea of an increasing time interval to repeat the process. He believes that is a good idea

because when a company starts to roll out this business continuity plan and they did not have such a plan before, many measures will have to be taken to reach a desirable situation. To ensure progress is being made to improve the situation, he argued repeating the plan every three months until the measures have been completed and the risks have decreased. After that, you increase the time interval to six or twelve months. The manager mentioned six months is a good starting point. Based on that, an organization can choose another time interval.

6 Discussion

As seen in the previous section, some conflicting opinions were given by the interviewees. The different roles and backgrounds the interviewees have could be the cause of these conflicting opinions. The owner, functional administrator, and manager are not involved with the technical part of the business as much as the technical administrator and the software developer are. However, the truck factor is most relevant to people who make business decisions, such as the owner, the manager, and partly the functional administrator. In this section, we will discuss why some decision-makers without technical knowledge are reluctant to use the truck factor and how what changes could to fix that. For this discussion, this section will have the same structure as section 5.

6.1 Truck factor usability

Opposing opinions were given by a few interviewees on whether they thought the truck factor would be valuable for the company. While the owner's suggestion of manually listing technical components and assigning knowledgeable individuals to each component is thorough and therefore might seem better, it is important to consider the potential drawbacks, particularly for SMEs. The proposed approach is undeniably time-consuming, requiring significant resources and effort. What makes this approach so time consuming is that As knowledge and personnel dynamics often change within companies, continuously updating and maintaining the manual component listings can become burdensome and impractical. This time investment translates into increased costs for the company, both in terms of personnel allocation and potential delays in addressing continuity risks. For SMEs with limited resources, such a detailed manual process may not be cost-effective or efficient [STB11]. Investing substantial time and effort into manually listing components might divert critical resources from other important operational aspects. It could potentially strain the company's productivity, hindering its ability to address immediate challenges and seize growth opportunities. Moreover, the dynamic nature of business environments necessitates flexibility and adaptability. The owner's proposed approach might not adequately address the evolving nature of continuity risks, because it makes maintaining accuracy of the continuity plan more difficult, which is one of the meta-requirements [JN+22].

In contrast to the listing method, leveraging the existing truck factor methodology provides a more efficient and scalable solution for SMEs. The truck factor offers a high-level assessment of knowledge concentration risks without requiring extensive manual efforts. By focusing on the overall risk level, SMEs can identify critical areas and allows management to allocate resources accordingly. This approach allows for more streamlined decision-making, ensuring that limited resources are deployed effectively to address continuity risks promptly.

To provide more detailed insights, an enhancement to the truck factor methodology could involve printing out all the files marked by the algorithm as abandoned. By expanding the truck factor's output to include the list of abandoned files, the level of specificity would increase. While giving the list of abandoned files as output could provide more detailed insights, it still might not be immediately useful to individuals without technical knowledge. File names often contain technical terms and references that are difficult for non-technical stakeholders to understand. To make this information more accessible, supplementary documentation, explanations, or visual aids

should accompany the list of abandoned files. These measures would help people in managerial roles grasp the functionality of the files and identify to which component it belongs. This would allow them to make more informed decisions with the help of the truck factor.

6.2 Algorithm Accuracy

As mentioned in the results, the algorithm provided incorrect estimates for repositories with incomplete git histories. The fact the algorithm does not work properly when this is the case is not surprising, because the algorithm is based entirely on the git history of a repository. When migrating a code base to Git, every file of the code base gets pushed into Git with one commit, from one person. This person is then the only author of each file in the repository, based on the git history, even though this person probably did not write all that code. This problem causes the truck factor to be inaccurate for organizations that did not start development in Git. That means code bases from before 2005 automatically will not have accurate truck factor estimates, since Git was invented in 2005.

We see two options to try and correct the inaccurate estimations. The first option is to add a feature to the truck factor which excludes files that were added in the first commit and were not touched since. The second option is to add a time range feature to the truck factor. This feature allows the user of the algorithm to set a range of time to execute the algorithm on, thereby excluding files that did not have any commits during the time range from the truck factor. Note that this option could also give inaccurate results when the time range is too small. Therefore, careful considerations have to be made on how to approach a situation such as the one with this particular repository.

6.3 Proposed measures

The participants were generally satisfied with the proposed measures, with some minor additions and remarks. Documentation was unanimously acknowledged as an effective approach for reducing continuity risks. The functional was especially concerned with having enough visuals in the documentation. Code reviews were seen as a suitable way to create a more active community, while the technical administrator suggested extending collaboration by discussing implementation with other developers before writing code. Automatic and well-named tests were considered essential, with the functional administrator valuing them even above documentation. Differing opinions arose regarding code comments, with the technical administrator valuing their role in explaining code rationale for future refactoring.

6.4 Additional measures

The two new measures that were proposed by the interviewees could be valuable additions if the original measures prove to be insufficient. The first measure involves hiring a third-party developer, who is available in times of crisis but does not cost a full salary. This measure doesn't increase the truck factor however and still requires the other measures. Therefore, this measure could only be an addition to the already existing measures. The second measure focuses on code structure, emphasizing its role in improving the debugging process and reducing the risk of prolonged

downtime. This measure would go hand in hand with code readability, so should be implemented simultaneously.

6.5 Process

Almost everyone agreed the process should be repeated regularly to detect changes in continuity risks and monitor whether progress is being made on repositories for which measures have been implemented. Only the functional administrator did not care about repeating the process. His skepticism was rooted in the fact that he did not like the use of the truck factor itself, because he did not want to implement it the first time he also did not want to implement it again after that. The different opinions about the timeline suggest that there is some room for flexibility on this front when implementing the design. Most people seemed fine with six months but did not have a strong opinion about it. We suggest repeating the process every three to twelve months, subject to adjustment based on experience. Also, customization and evaluation of repetition timelines based on organizational characteristics and risk profiles are vital. The benefits of continuous improvement and adaptability should be weighed against resource constraints. In summary, repetition of the process is generally supported, and an appropriate repetition timeline should be tailored to the organization's needs and capabilities.

6.6 Limitations

Some critiques given in the interviews were tempting us to do another iteration of the designing process. In this iteration, we could have implemented the possible solutions to the problems some interviewees had with the design. After implementing these solutions, the updated design would have been evaluated again with the same group. However, some participants in the interview did not appear open to this idea. They inherently disliked the truck factor, so we reasoned another iteration would not give insightful feedback. The fact only iteration was done limits this study because another iteration of the design might have given different results and thus a different conclusion.

7 Conclusions and Further Research

In this bachelor thesis, we researched how valuable the truck factor is for small to medium-sized software development enterprises to incorporate into their business continuity management. We have conducted this research as a case study with a Dutch software development company. To test the applicability of the truck factor to their business we created a design to implement it into their business continuity management. To start the design we picked the most accurate and accessible truck factor algorithm currently available in the literature, which proved to be the algorithm made by Avelino et al. [APHV16]. The algorithm estimates the truck factor based on the Git commit history of a repository. Based on the truck factor and the business value of a repository, we proposed a risk score matrix. Risk scores indicated whether measures should be taken to reduce the continuity risks for that repository. The measures we proposed to reduce continuity risk were: shift employees to projects with a lower truck factor or hire new developers to increase the truck factor. Secondary measures were: write sufficient documentation, create a more active community by doing code reviews, run automatic unit- and integration tests, and make sure code is easily understandable by having code comments using good coding practices [APHV16]. In the entire design, we kept meta-requirements and processes of SME BCP development in mind [JN+22].

To evaluate the design we interviewed 5 employees of the Dutch software company. Three people were positive about the design, they were either people close to the development process or were developers themselves. However, two people had major issues with the truck factor's level of detail. The output the truck factor provided was not useful to them as people in managerial roles. Because the management team has to make decisions on how to manage continuity, the truck factor must be understandable and informative for them. Another issue we ran into with the truck factor during the evaluation is that the truck factor algorithm gives inaccurate results for repositories that were migrated to Git later on in their development. To conclude, the truck factor in its current form is a slightly insufficient to implement for software development SMEs. It is already useful for larger companies, since continuity risks are less obvious for them. However, for the truck factor to be practically useful for SMEs the lack of detail has to be solved by giving the abandoned files as an output of the algorithm.

In further research, the two issues with the truck factor could be (partially) solved. The lack of detail in the truck factor's output could be explained in more detail by giving the abandoned files as output. Ideally, the abandoned files are shown in a graphical interface with an explanation of their functionality. With that information, the management team can make more informed decisions based on the truck factor. The inaccurate results in repositories that were migrated to Git could be made more accurate by excluding the first commit from the algorithm. Even though not everyone approved of the truck factor, the interviewees did reach a consensus on the measures we suggested. Only some minor additions were given, which could be implemented on top of the existing measures. With the two additions to the truck factor we just suggested, further research can make the truck factor practically valuable for SMEs. In further research, the adjusted design could be evaluated on multiple organizations, to get more extensive results and understand the practical applicability of the truck factor even better.

References

[ANLV+20] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. Software documentation: the practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 590–601, 2020.

[APHV16] Guilherme Avelino, Leonardo Passos, Andre Hora, and Marco Tulio Valente. A novel approach for estimating truck factors. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016.

[BR93] Sam Blili and Louis Raymond. Information technology: Threats and opportunities for small and medium-sized enterprises. *International Journal of Information Management*, 13:439–448, 1993.

[BR15] Gabriele Bavota and Barbara Russo. Four eyes are better than two: On the impact of code reviews on software quality. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 81–90. IEEE, 2015.

[CIC15] Valerio Cosentino, Javier Luis Cánovas Izquierdo, and Jordi Cabot. Assessing the bus factor of git repositories. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 499–503. IEEE, 2015.

[FMH07] Thomas Fritz, Gail C Murphy, and Emily Hill. Does a programmer's activity indicate knowledge of code? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 341–350, 2007.

[FVF17] Mivian Ferreira, Marco Tulio Valente, and Kecia Ferreira. A comparison of three algorithms for computing truck factors. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, May 2017.

[Her10] Brahim Herbane. Small business research: Time for a crisis-based view. *International small business journal*, 28(1):43–64, 2010.

[Her19] Brahim Herbane. Rethinking organizational resilience and strategic renewal in smes. *Entrepreneurship & Regional Development*, 31(5-6):476–495, 2019.

[JETK22] Elgun Jabrayilzade, Mikhail Evtikhiev, Eray Tüzün, and Vladimir Kovalenko. Bus factor in practice. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 97–106, 2022.

[JN+22] Jonna Järveläinen, , Marko Niemimaa, Markus P. Zimmer, and and. Designing a thrifty approach for SME business continuity: Practices for transparency of the design process. *Journal of the Association for Information Systems*, 23(6):1557–1602, 2022.

[KS18] Hanna Kinnunen and Mikko Siponen. Developing organization-specific information security policies by using critical thinking. In *Pacific Asia Conference on Information Systems*. Association for Information Systems, 2018.

[Nat15] United Nations. Sendai framework for disaster risk reduction 2015-2030, 2015.

[Ola03] Michael Olan. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*, 19(2):319–328, 2003.

[PTG+20] Ken Peffers, Tuure Tuunanen, Charles E Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, and Johanna Bragge. Design science research process: A model for producing and presenting information systems research. *arXiv preprint arXiv:2006.02763*, 2020.

[RMT11] Filippo Ricca, Alessandro Marchetto, and Marco Torchiano. On the difficulty of computing the truck factor. In *Product-Focused Software Process Improvement: 12th International Conference, PROFES 2011, Torre Canne, Italy, June 20-22, 2011. Proceedings 12*, pages 337–351. Springer, 2011.

[RZDM16] Peter C Rigby, Yue Cai Zhu, Samuel M Donadelli, and Audris Mockus. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya. In *Proceedings of the 38th International Conference on Software Engineering*, pages 1006–1016, 2016.

[SSSG+10] David J Storey, George Saridakis, Sukanya Sen-Gupta, Paul K Edwards, and Robert A Blackburn. Linking hr formality with employee job quality: The role of firm and workplace size. *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in Alliance with the Society of Human Resources Management*, 49(2):305–329, 2010.

[STB11] Bridgette Sullivan-Taylor and Layla Branicki. Creating resilient smes: why one size might not fit all. *International Journal of Production Research*, 49(18):5565–5579, 2011.

[TRM11] Marco Torchiano, Filippo Ricca, and Alessandro Marchetto. Is my project's truck factor low? theoretical and empirical considerations about the truck factor threshold. In *Proceedings of the 2nd international workshop on emerging trends in software metrics*, pages 12–18, 2011.

[Uni03] European Union. Comission recommendation of 6 may 2003 concerning the definition of micro, small and medium-sized enterprises, 2003. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32003H0361&from=NL.

[Vos98] Robert W Vossen. Relative strengths and weaknesses of small firms in innovation. *International small business journal*, 16(3):88–94, 1998.

[VS11] John Vargo and Erica Seville. Crisis strategic planning for smes: finding the silver lining. *International Journal of production research*, 49(18):5619–5635, 2011.

[VvGMW16] Stefan Voigt, Jörg von Garrel, Julia Müller, and Dominic Wirth. A study of documentation in agile software projects. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–6, 2016.

[ZSK+10] Nico Zazworka, Kai Stapel, Eric Knauss, Forrest Shull, Victor R Basili, and Kurt Schneider. Are developers complying with the process: an xp study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.