

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Continuous assessment of code quality through software analytics in a start-up environment

Duarte Oliveira



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Filipe Correia

Second Supervisor: Eduardo Guerra

July 25, 2021

Continuous assessment of code quality through software analytics in a start-up environment

Duarte Oliveira

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2021

Abstract

Software practitioners need to continuously assess code quality to increase the quality of their projects and avoid software problems that could cost a large amount of money and resources later in development. This helps to minimize resources and costs for the company, which is very significant for start-up companies. Start-ups face big risks and uncertainty and need to adapt quickly to market changes to maintain their business. More often than not, they need to ignore the quality of the code to produce new content or make significant business changes. This phenomenon is also known as technical debt.

The use of analytics in software data has become increasingly popular. However, analytics tools setup can be complicated and time-consuming to practitioners. The large amount of raw data generated by analytics tools can be another obstacle since it requires practitioners' skills to gain insight from data with practical value. A large amount of raw data can limit the ability to understand the main problems regarding code quality and the metrics established. This also makes it challenging to assess technical debt when making significant changes and producing value for the company.

Approaches to software analytics have taken multiple forms. Software Quality models can give us the definition of quality, but it doesn't provide a way for the teams to adopt software analytics practices. There is a need to operationalize software quality models to give the teams a clear and structured method to assess code quality. Assessment approaches focus on finding specific metrics that should be used to evaluate code quality or outline activities in a structured way for a team to follow to assess the system's quality. However, it doesn't provide a straightforward way to act upon these findings. Continuous improvement approaches like the Software Analytics Canvas offer us a framework for finding the issues, finding the correct metrics for these issues, assessing them, and continuously improving the system's quality.

This work aims to validate the approach called the Software Analytics Canvas in a start-up environment. We report an industrial case study on the two software teams of a start-up where we assess the usefulness and ease-of-use of the approach, the improvement of the awareness on the state of quality of their system, and whether the approach could effectively improve the quality of their system. The teams used the approach for five sprints, which is equivalent to ten weeks of development. We carried out several surveys and collected several metrics to support answering our research questions. It helped us confirm that the approach was fairly easy to understand, and although harder to use than they expected, it was still beneficial. The participants also confirmed that they felt more aware of the system's quality as it helped find several issues they did not know existed. And although we can't know the overall improvement of the quality of their system, the metrics that they collected had positive improvements and will continue to improve in the future.

Keywords: software teams, software practitioners, start-ups, software analytics, raw software data, code quality, software analytics canvas

Resumo

Os profissionais de software precisam de avaliar continuamente a qualidade do código para aumentar a qualidade de seus projetos e evitar problemas de software que podem custar muito dinheiro e recursos posteriormente no desenvolvimento. Isto ajuda a minimizar recursos e custos para a empresa, que é muito significativo para start-ups. As start-ups enfrentam grandes riscos e incertezas e precisam de se adaptar rapidamente às mudanças do mercado para manter os seus negócios. Maior parte das vezes, eles precisam ignorar a qualidade do código para produzir novo conteúdo ou fazer mudanças significativas no negócio. Este fenômeno também é conhecido como dívida técnica.

O uso de analytics em software está-se a tornar cada vez mais popular. No entanto, a configuração das ferramentas de analytics pode ser complicada e demorada para os profissionais. A grande quantidade de dados brutos gerados por ferramentas de analytics pode ser outro obstáculo, pois requer habilidades dos profissionais para obter informação útil a partir dos dados. Uma grande quantidade de dados brutos pode limitar a capacidade de entender os principais problemas relacionados com a qualidade do código e as métricas estabelecidas. Isto também se torna um desafio quando se tem de avaliar a dívida técnica ao fazer mudanças significativas e gerar valor para a empresa.

As abordagens de software analytics assumiram várias formas. Os modelos de qualidade de software podem nos dar a definição de qualidade, mas não fornecem uma maneira para as equipas adotarem práticas de software analytics. É necessário operacionalizar modelos de qualidade de software para fornecer às equipas um método claro e estruturado de avaliação da qualidade do código. As abordagens de avaliação concentram-se em encontrar métricas específicas que devem ser usadas para avaliar a qualidade do código ou delinear atividades de uma forma estruturada para uma equipa avaliar a qualidade do sistema. No entanto, não fornece uma maneira direta de agir com base nessas descobertas. Abordagens de melhoria contínua, como o Software Analytics Canvas, oferecem-nos uma estrutura para encontrar os problemas, encontrar as métricas corretas para esses problemas, avaliá-los e melhorar continuamente a qualidade do sistema.

Este trabalho tem como objetivo validar a abordagem chamada Software Analytics Canvas num ambiente de start-up. Nós relatamos um estudo de caso industrial com as duas equipas de software de uma start-up, onde avaliamos a utilidade e facilidade de uso da abordagem, a melhoria da noção sobre o estado da qualidade do sistema e se a abordagem poderia melhorar efetivamente a qualidade do mesmo. As equipas usaram a abordagem durante cinco sprints, o que equivale a dez semanas de desenvolvimento. Realizamos vários questionários e recolhemos várias métricas para ajudar a responder às nossas perguntas de investigação. Isto ajudou-nos a confirmar que a abordagem era bastante fácil de entender e, embora mais difícil de usar do que eles achavam, ainda assim foi benéfica. Os participantes também confirmaram que se sentiam mais conscientes da qualidade do sistema, pois ajudou a encontrar vários problemas que eles não sabiam que existiam. E embora não possamos saber se a qualidade geral do sistema melhorou, as métricas que foram

recohidas tiveram melhorias positivas e continuarão a melhorar no futuro.

Keywords: equipas de software, profissionais de software, start-ups, software analytics, dados brutos, qualidade de código, software analytics canvas

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Objectives and Contributions	2
1.4	Document Structure	2
2	Background	3
2.1	Software quality	3
2.2	Software Analytics	4
2.3	Start-ups and Technical Debt	4
2.4	Chapter Conclusions	4
3	Related Work	7
3.1	Software Quality Models	7
3.1.1	The Quamoco Quality Modeling Approach	8
3.1.2	Q-Rapids Quality Model	9
3.1.3	Discussion	10
3.2	Software Analytics Assessment Approaches	10
3.2.1	Goal, Question, Metric	11
3.2.2	SCQAM: Scalable Structured Code Quality Assessment Method	11
3.2.3	Discussion	12
3.3	Continuous Improvement Approaches	13
3.3.1	PDCA: Plan, Do, Check and Action	13
3.3.2	Software Analytics Canvas	13
3.3.3	Discussion	16
4	Problem statement and Proposed Solution	19
4.1	Scope	19
4.2	Research Questions	20
4.3	Methodology	20
5	Industrial Case Study	23
5.1	Goals	23
5.2	Study Design	24
5.3	Results Analysis	26
5.3.1	Company Background in Software Analytics	26
5.3.2	Usage of The Software Analytics Canvas	27
5.3.3	Usefulness and Ease-of-use Perception	30

5.3.4	Code Quality Awareness	34
5.3.5	Code Quality Improvement	36
5.4	Challenges and Possible Improvements	38
5.5	Threats to Validity	39
6	A New Observed Pattern: Embedded Improvements	41
6.1	Problem	41
6.2	Solution	42
6.3	Consequences	42
6.4	Example	43
7	Conclusions	45
7.1	Summary	45
7.2	Research Answers	46
7.3	Future Work	47
A	Surveys	49
A.1	Kick-off Survey	49
A.2	Sprint Survey	54
A.3	Final Assessment Survey - Tech	60
A.4	Final Assessment Survey - Managers	68
	References	73

List of Figures

3.1	Quamoco meta-model diagram	9
3.2	Software Analytics Canvas Version 1.0	14
3.3	Software Analytics Canvas Version 2.0	16
5.1	Study Design Diagram	25
5.2	Software Analytics Canvas Version 2.1 Issue Section	28
5.3	Software Analytics Canvas Version 2.1 Tasks Section	28
5.4	Software Analytics Canvas lane usage per sprint	29
5.5	Software Analytics Canvas Lane Understanding per Sprint	32
5.6	Software Analytics Canvas Lane Ease-of-use per Sprint	33
5.7	Software Analytics Canvas Usefulness Perception per Sprint	34

List of Tables

5.1	Responses to the background survey	26
5.2	Software Analytics tasks to Regular Tasks comparison by sprint	29
5.3	Software Analytics Retrospectives to Regular Retrospectives comparison by sprint	30
5.4	Average of responses to the sprint survey regarding understanding and ease-of-use of the Software Analytics Canvas	31
5.5	Average of responses to the sprint survey regarding the usefulness of the approach	33
5.6	Responses to the final survey to the managers regarding usefulness, understanding and ease-of-use	35
5.7	Responses to the kick-off survey regarding the code quality awareness	35
5.8	Average of responses to the sprint surveys regarding the code quality awareness .	35
5.9	Responses to the final survey regarding the code quality awareness before the Canvas	36
5.10	Responses to the final survey regarding the impact on code quality awareness . .	36

Chapter 1

Introduction

1.1 Context

Software companies have multiple on going projects. Assessing the quality of these products helps the team keep the quality at a good level. Having a good quality software system makes it easier for the team to maintain a software system and develop new new features.

To assess code quality, software teams can use software analytics, which takes into account source code, static and dynamic characteristics of a software system, and also development processes. The goal of software analytics is to provide insightful and actionable information to help software teams make decisions and accomplish software tasks like refactoring while knowing the impact of the changes [25].

This is especially useful for companies like start-ups. These companies face very high business uncertainties and need to adapt quickly [19]. With very rapid changes, comes the necessity of quick reactions and decisions, and assessing the technical debt that a change will generate becomes extremely valuable for these companies.

1.2 Motivation

Companies generally have multiple critical projects with very high impact on their business and they want to avoid discovering software quality problems when the costs of fixing them are already very high. Assessing code quality and keeping it at a stable level helps the organization save resources and helps decreasing the cost of building and maintaining software. Specially in start-ups, this becomes extremely useful since they need to evolve fast to respond to market changes.

Furthermore, in code quality decision-making processes, intuition is often used by developers based on their knowledge and past experiences instead of using facts based on the project itself. Software analytics helps eliminating the intuition factor by providing real insightful and actionable data that can help in identifying measures to be taken. With these insights, the team can focus on improving the code quality and immediately know the impact of these changes.

However, when assessing code quality, software teams use automated tools such as SonarQube to help gather data about a code-base, including numerous software metrics and the detection of bugs and code smells. Teams employ a great effort in setting up these tools, but are at a loss when dealing with the large amounts of data that they produce.

There is also a lack of consolidated approaches on how to introduce software analytics concepts and practices into an agile development context [5]. Continuous improvement approaches help the team plan and take action on improving the code quality of their projects. The Software Analytics Approach that will be further analyzed in Section 3.3.2 is an example of a systematic approach that works well with Agile teams. Approaches like the Software Analytics Canvas need to be explored and verified.

1.3 Objectives and Contributions

The goal of this work is to study the adoption of data-driven code quality assessment practices in software development, and conduct an industrial case study using a systematic approach like the Software Analytics Canvas [5]. This dissertation is expected to help the participating software teams in the field to adopt data-driven practices for improving code quality, but evaluate the Software Analytics Canvas technique in a start-up environment.

The outputs of this thesis are as follows.

- State of the art review and analysis of related work.
- An industrial case study by applying the Software Analytics Canvas approach in a start-up environment.
- Define a new pattern for implementing software analytics in small teams.
- Generate possible alterations to the Software Analytics Canvas approach.

1.4 Document Structure

In the next chapter, we will cover some background concepts regarding software quality, software analytics, start-ups and technical debt. In Chapter 3, we will cover in greater detail some software quality model, some software quality assessment approaches and some continuous improvement approaches. In Chapter 4, we describe the scope of the problem, the research questions, the methodology established and the work planned for the next phase. In Chapter 5, we will describe the industrial case study conducted on a start-up by analyzing the goals, design of the study, analyzing the results, describing a new pattern found with this study and some challenges and possible improvement to the studied approach. Chapter 7 provides a quick overview of the work done, provides clear answers to the research answers defined and explores aspects for future work.

Chapter 2

Background

This chapter covers the topics needed to understand this dissertation fully. In Section 2.1 we explain the concept of software quality. In Section 2.2, we examine concepts related to software analytics, which are the main focus of this document. And finally, in Section 2.3, we explore the concept of a startup and the notion of technical debt.

2.1 Software quality

Software quality can be classified in two different ways. The first one is how well the produced software reflects the functional requirements defined and how well it compares to other competitor products. The second definition is related to non-functional requirements and is the one we refer to in this document. It takes into account several abstract quality characteristics called factors. It's more about the code-base itself, and there are two types of factors: internal and external. Users care more about the external factors. However, developers need to be careful about internal factors as well [23].

External factors are related to the usefulness perceived from an external point of view. External factors are more important for the users since they can be perceived by using the product. These factors include functionality, reliability, usability, efficiency, flexibility, simplicity, and others [23].

Internal factors need to be assessed as well. These factors are crucial for developers since they impact the development process. It helps avoid high costs of production and discovering bugs early in development. These factors include maintainability, portability, reusability, testability, and others [23].

Although maintaining the quality of a software product can be costly, for example, investing in developers' training with methodologies, review meetings, and other activities, not investing in it can be even more expensive. Quality improvements can result in cost savings that outweigh the efforts of training and evaluating the code [22].

2.2 Software Analytics

Software development activities can produce large amounts of data from the source code, test reports, bug reports, etc. As stated by Zhang et al. [25] Software analytics leverages these information sources to obtain insightful and actionable information for developers, managers, and product owners to make decisions and accomplish development tasks. The ultimate goal is to improve the productivity of development activities, user experience, and software quality. Although in this document we only focus on the improvement of code quality, analytics can measure products, features, or quality attributes [9].

The users related to these activities are, first and for most, developers who participate in planning, setting up tools, gathering metrics, and acting on the code quality. Some companies also have full tester teams that also take advantage of testing metrics to improve the code base's testability further. But software managers are also practitioners since they have an essential role in the decision-making process [25].

The main types of technologies used for software analytics are large-scale computing used for large-scale datasets, information visualization tools for data analysis and viewing results, and machine learning for insight analysis [25].

Several approaches exist to help teams adopt software analytics practices in their development process. The Software Analytics Canvas [5] is the one we are going to be using in the case study of this work and will be further explained in Section 3.3.2.

2.3 Start-ups and Technical Debt

Start-ups are companies or projects created to develop and validate a scalable business model. These companies are trying to find a not easy solution, and success is not guaranteed [18]. Unlike entrepreneurship, start-ups are businesses that have the desire to grow beyond just a project and want to become a large scale business with multiple founders [21]. Start-ups usually face high levels of uncertainty, often have frequent changes of direction, and face high rates of failure [20].

Like financial debt, technical debt [7] is the extra effort to maintain and develop a product due to sub-optimal or flawed solutions. It occurs mainly when a software team takes quick shortcuts by delivering faulty or unverified solutions to achieve faster time-to-market. The debt should be re-paid quickly by fixing the flawed implementations. Debt can be produced due to various reasons and might not always be due to intentional reasons. Other causes can be ignorance of good practices, developers with a lack of skills, oversight, or even pragmatism [14].

2.4 Chapter Conclusions

Software quality is the central focus for efficient and effective maintenance and development of a product. When quality is ignored, a product's technical debt keeps rising, and the debt gets harder to pay. For start-ups that need quick time-to-market, they must start using techniques to assess

their products' quality to reduce the potential future technical debt. Even if a decision is made to take shortcuts to produce value quickly, the quality should always be monitored to understand such shortcuts' impacts.

This is where software analytics matters. Software quality attributes can be measured with analytics extracted from the code base and development processes. In the next chapter, we will understand how quality is defined and how to measure it efficiently.

Chapter 3

Related Work

This chapter covers the state of the art regarding software analytics models and approaches, with particular attention to the work already done on the Software Analytics Canvas and relevant technologies used in this area. Section 3.1 introduces several software quality models. In Section 3.2, we describe some software quality assessment approaches. Section 3.3 discusses some continuous improvement approaches where we will further talk about the work already done on the Software Analytics Canvas. And, finally, in Section 3.4, we will discuss some relevant technologies used in the area.

3.1 Software Quality Models

Software Quality Models are a standardized way of measuring a software product. These models provide a framework for defining quality attributes, building and measuring software products. With several software systems built every day, a rise in the need for ensuring the quality of these products was noticed. These models have been around for a long time now, and the development can be backdated to the 1970s.

One of the first models was McCall's (1977) [16]. The model was developed to assess relationships between external factors and product quality criteria. It defines product quality under three perspectives: user's perspective, developer's perspective, and metrics, which are defined and used to provide a scale and measurement method. Under these three perspectives, the factors are organized in a hierarchy. The factors were reduced to 11 to understand the model better. The factors that define the product's quality are Correctness, Reliability, Efficiency, Integrity, Usability, Maintainability, Testability, Flexibility, Portability, Reusability, and Interoperability. The only thing this model doesn't contribute to is directly on the functionality of the product.

Boehm's model (1978) [3] was introduced after and proposes a hierarchy of quality characteristics. The three primary characteristics are utility, maintainability, and portability. They are then decomposed in several other levels until primitive characteristics are reached. This model's main elements are quality characteristics, sub-quality characteristics, primitive quality characteristics, and quality metrics.

After several models were defined, ISO/IEC JTC1 decided to develop a consensus on software quality and created the ISO 9126 quality model [12] to encourage standardization worldwide. It is a tree hierarchy model with three main types of quality: internal quality, external quality, and quality in use. The model specifies the following characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. The characteristics are then divided into 21 other sub characteristics, which are manifested in metrics that can be measured.

In the next two sections, we will analyze in greater detail a quality meta-model and another quality model applied to a tool for decision support.

3.1.1 The Quamoco Quality Modeling Approach

Quamoco [24] is a modeling approach designed to bridge the gap between models that provide abstract quality attributes and models that provide concrete quality assessments. Quality models can be categorized according to two types. The first type of quality model describes and structures general concepts that constitute high-quality software but lacks the ability to be used for actual quality assessment or improvement. The second type of quality model is too tailored for specific domains or single aspects of software quality, which misses the connection to higher quality goals and makes it difficult for developers and stakeholders to understand the importance of quality problems.

For the Quamoco modeling approach, a meta-model was defined to better understand the relation between a model's concepts. At the center of this meta-model resides the factor which expresses a property of an entity. This factor can either be a quality aspect, representing an abstract quality goal, or a product factor, which is a measurable attribute of the product. Both can be refined and produce their acyclic graph. Also, a product factor impacts a quality aspect. This represents the way factors relate to each other. A factor is always associated with an entity, and it describes a property of the entity. The name of this property is the factor's name. A factor also has an evaluation associated with it, which can be a specific measure in the case of a product factor. Still, the assessment can again come from the evaluation of the sub-factors. A measure is not restricted to only one product factor and can be measured from multiple instruments. Instruments allow collecting measures from, for example, static analysis tools.

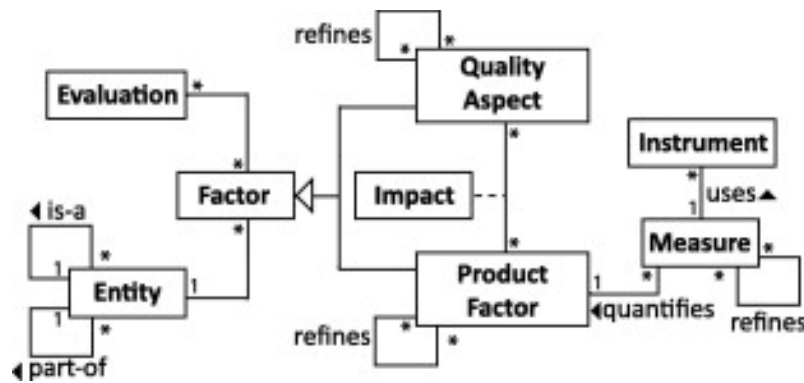


Figure 3.1: Quamoco meta-model diagram, by Wagner et al. [24]

A base model was defined from this meta-model to define software quality in a way that is easy to implement in a tool-supported quality assessment method and is accessible to multiple products. The model comprises a wide selection of factors, more precisely, 286 factors of 112 entities. From these factors, 202 define impacts on quality aspects, which makes 492 impacts, and 192 generated specific measures, which makes a total of 526 measures and 542 instruments of measure.

3.1.2 Q-Rapids Quality Model

The Q-Rapids quality model [15] was created based on the information acquired in workshops with four companies. This model was generated for assessing every quality aspect in the form of one single tool. The model analyses heterogeneous data sources during development and at run-time and generates quality alerts when a specific element is at risk. The model considers several aspects such as quality aspects, product and process factors, assessed metrics, and raw data. These product and process factors are related to maintainability, reliability, functional suitability, and productivity.

Several products and process factors constitute the Q-Rapids quality model. There are five product factors and one process factor.

- **Code quality** (product factor): refers to the maintainability of the resources and includes complexity, comment density, and code duplication
- **Blocking code** (product factor): refers to metrics regarding the fulfillment or violation of quality rules and technical debt issues
- **Testing status** (product factor): refers to the test coverage collected from the static code analysis
- **Software stability** (product factor): refers to the crashes at run-time
- **Software usage** (product factor): refers to the number of times a feature is used

- **Issue's velocity** (process factor): refers to the comparison between the estimated time and the invested time into an issue

The assessment of the Q-Rapids quality model follows the Quamoco bottom-up approach referred to in the last section. It starts by first gathering raw data from heterogeneous data sources. The basic metrics are identified, which leads to the interpretation of these metrics with a utility function using preferences and judgments of experts or learned data (i.e., machine learning). The output of these utility functions are valued between 0 and 1, 0 being the worst. The assessed metrics are then aggregated into product and process factors, which are then aggregated into quality aspects that can be easily interpreted.

The implementation of the model comes in the form of a tool. The Q-Rapids tool supports data gathering from static code analysis, tests executed in development, code repositories, and issue tracking tools, consisting of the data producer layer. The data ingestion layer consists of several Apache Kafka connectors to gather data from the producers. The third layer is the distributed data sink layer used for data storage, indexing, and analysis. And last but not least, the data analysis and processing layer perform the quality model assessment. It queries the data sink layer and applies the utility functions to interpret the raw data.

3.1.3 Discussion

The source code is the central artifact of a software project. Software quality models help us define what we need to measure and interpret it to understand the quality of a specific product. Like McCall's model, some models were made for a particular environment, others to standardize quality like the ISO 9126, but others tried to generalize by creating meta-model (e.g., Quamoco), which is a framework for building quality models.

Most quality models do an excellent job describing properties that relate to software quality but don't quite provide a way to implement it in a project, measure these properties, and aggregate the findings into an overall quality rating. Some projects like the Quamoco project have been trying to bridge the gap between these abstract quality concepts and the specific metrics. And projects like the Q-Rapids and the Quamoco project created tools to operationalize the model itself.

Although models are useful for defining quality, there is still the need for assessment approaches for organizations to implement in their projects. In the next section, we address some software quality assessment approaches that exist today.

3.2 Software Analytics Assessment Approaches

In this section, we detail some software analytics assessment methods. These methods provide actionable data that software teams can interpret in order to improve the quality of software. It's a way to operationalize software quality models. Manual assessments are too effort extensive and time consuming. Software or assessment teams need a more systematic and automated way to evaluate source code and provide insights to further improve code quality.

3.2.1 Goal, Question, Metric

The Goal Question Metric [4] is an approach that states that for an organization to measure the quality of a software product, it must first specify the goals, trace those goals to the data (source code static analysis, dynamic analysis, etc.) and then provide a framework to interpret the data in correlation to the defined goals. This approach was initially defined to evaluate a set of projects in the NASA Goddard Space Flight Center environment but has expanded to a bigger context. It is a setting step in the Quality improvement paradigm [2].

This approach has three important levels that we must consider. The first one is the **conceptual level** or the goals. A goal is defined in relation to a quality model, a specific environment, multiple points of view, and objects measured. These objects of measurement can be products (i.e., artifacts, deliverables, documents), processes (i.e., software-related activities), and resources (i.e., items used by processes) of an organization.

The second level is the **operation level** or the questions. This level is characterized by a set of questions that categorizes an object of measurement in relation to the quality issue chosen and attempts to determine the object's quality from the selected viewpoint.

The third level is the **quantitative level** or the metrics. The metrics are a set of data used to respond to the question and, therefore, quantify the specified goal. When interpreting the data, there are two possible outcomes. Either the data is objective and depends only on the object being evaluated, or it is subjective and depends on the object and on the viewpoint and environment in which the object exists.

The approach itself starts by defining the goals divided into several questions, which will be answered with metrics collected from the data available. The goals are based on three primary sources of information. The first is the organization's policy and strategy, from which we will be able to take the issue and purpose of the goal. The second is the description of the process and products from which we obtain the objects of measurement. The third source is the organization's model, which can provide us with the viewpoints of the organization. From the specification of the goals, we can start to derive the questions that will characterize the goals in a quantifiable way. Once we have defined the questions, we can begin collecting the appropriate metrics to answer those questions. To choose these metrics, several factors must be considered, like the amount and quality of the data to make sure that the data being used is reliable. Other factors are the maturity of the objects and measurement and the learning process that the approach requires.

3.2.2 SCQAM: Scalable Structured Code Quality Assessment Method

SCQAM (Structured Code Quality Assessment Method) [11] is a tool-assisted and expert-based quality assessment method developed by Siemens. The goal was to have a scalable approach to identify issues that have a negative impact on the code quality of projects across their organization. This method is derived from the EMISQ [17], a quality model-driven method. The EMISQ method, which was being used before, was too effort-intensive and time-consuming. Scaling the previous process for the hundreds of projects they had posed many challenges.

SCQAM method comprises three phases: the input phase, the processing phase, and the output phase. In the **input phase**, the assessors take access to code artifacts like code repository, coding guidelines, used libraries, built binaries, and build environment to start the processing phase.

In the **processing phase**, there are three parts. The first one is the **preparation** where the software team provides the project information to the assessors following a template. The components are then selected based on criticality, importance to the business, and stability of the component. This step is sometimes ignored if the project is small enough. Based on the selected components, criteria for assessment are established. And the last step is to produce an evaluation plan. The second part is the **measurement**. In this part, the assessors configure tools, run tools and collect results and reports. The last part is the **assessment** where tool reports are analyzed, and some manual code inspections can occur. The findings are then consolidated, aggregated, and documented.

In the **output phase**, the findings are communicated to the software team. They are given the documented results, a presentation about the most critical findings, reports, and code analysis tool outputs. In this phase, assessors and team members validate findings, create actionable items and collect feedback.

This method simplifies activities from the previous process by using review tools, automation of activities, report generation, and only the critical issues are considered for assessment.

3.2.3 Discussion

Software analytics assessment methods provide us with a way to assess software projects' code quality in a structured process. Some approaches like the Goal, Question, Metric focus more on finding the goals that the team wants to achieve, much like the Software Analytics Canvas, that we will explain in Section 3.3.2. By providing the goals first, the team can understand why the issue exists and the metrics being monitored have a purpose.

The SCQAM method is more focused on scaling the process of software quality assessment. It also comes from the company's necessity that created it since they were using techniques that took too much time and effort to implement. By the end of the assessment, many other problems could occur, and the technical debt generated could be a problem.

The SCQAM process is also more focused on process management and tool usage, while the Goal, Question, Metric is a lot more focused on establishing goals and which metrics should be used. Since the objective of SCQAM is scaling, this part is more automatic, and only the most critical issues are considered because of the number of projects being monitored.

These methods are useful for establishing metrics and how to perform the assessment. Still, continuous improvement methods should be considered. Assessment methods don't provide what to do with the information obtained and how to improve the code based on the actionable data acquired.

3.3 Continuous Improvement Approaches

Data-driven continuous improvement can help agile teams to save resources and decrease the cost of building and maintaining software. Software analytics can support agile teams to make more appropriate changes based on actual data, rather than only on their personal experiences or intuitions. Especially for start-ups where they can't have a full team focused on quality improvement, it is essential that approaches like these are used. In this section, we discuss some software analytics continuous improvement approaches but also one approach not directly related to code quality improvement but that could also be beneficial to analyze.

3.3.1 PDCA: Plan, Do, Check and Action

PDCA [13] is an acronym for Plan, Do, Check, and Action. It's a software quality continuous improvement approach. The approach is a cycle of four phases that forms a self-feedback system to improve the software project's quality. Every time a cycle is completed, the quality improves. The four stages of the PDCA approach are, as the name implies, Plan, Do, Check, Act.

In the **Plan** phase, the team defines the principles and aims and formulates the activities. The Plan phase requires the team members to answer six questions: what, why, who, when, where, and how. It also requires them to find problems, reasons behind these problems, define quality principles, aims and intentions. When the issues are confirmed, two types of measures are planned. The first one is emergency or temporary measures that exist to eliminate the problem. The second is permanent measures that exist to prevent problems from reoccurring.

In the **Do** stage, the team puts the plan into practice. It is when the team implements the practical measures and controls the process to reach the planned goal. Due to the planning phase being so thorough, the plan should be strictly followed. Some adjustments can be made if the original measures can't be implemented, but the adjusted measures should correspond to the goals defined.

The **Check** phase is the process of continuously collecting data, getting information from the implemented measures, and analyzing the results. This stage is crucial for self-improvement. It's where the team can find problems and improvement opportunities.

Finally, the **Action** stage is where the improvements are made based on the results and information acquired from the last phases. The issues are addressed to prevent the reoccurrence of problems. The PDCA cycle favors careful planning to take action.

3.3.2 Software Analytics Canvas

Software Analytics Canvas [5] is an approach to introduce software analytics concepts and practices in an agile development context. It's a goal-focused approach, much like the Goal, Question, Metric approach described in the previous section. The canvas is a hub of information flow related to software analytics projects where the team can plan their software analytics related activities. It's a situation awareness channel for all team members. The objective is to ensure that the teams

participate in all planning activities from the definition of the goals to the metrics defined and that the team understands the metrics being collected and that these metrics are articulated with the measurement goals. It's also useful to avoid monitoring unnecessary measurements since everything is goal-focused.








SOFTWARE ANALYTICS CANVAS												
PROJECT: <i>PROJECT NAME</i>			VERSION: <i>1.0</i>									
KEY ISSUES  <p>What do we want to know about the software, process and/or usage patterns?</p>	DATA SOURCES  <p>What are the data sources that can provide information on the issues raised?</p>	DATA GATHERING  <p>How to collect and analyze software data related to this issue?</p>	INSIGHTS  <p>What have we found out from the analysis?</p>	QUALITY THRESHOLDS  <p>What parameters we can establish to evaluate the quality of our decisions?</p>								
ANALYTICS IMPLEMENTATION  <p>How to implement software analytics tasks along with other tasks?</p> <table border="1"> <thead> <tr> <th>To Do</th> <th>DONE</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	To Do	DONE			INCREMENTAL GOALS  <p>What incremental goals can we establish based on the insights from data analysis?</p> <table border="1"> <thead> <tr> <th>To Do</th> <th>DONE</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> </tbody> </table>		To Do	DONE				
To Do	DONE											
To Do	DONE											

Figure 3.2: Software Analytics Canvas Version 1.0 [5]

The Canvas was built on several patterns mined using a systematic mapping study performed to investigate current software analytics trends for decision-making in software development [6, 5]. There's a total of eight patterns.

The first pattern is "**What You Want to Know**". It's related to the start of the approach, where the team members should define specific goals and issues to focus on. Decisions are usually made based on the developer's intuition, so the objective is to choose the problems that the team does not have information on to make these decisions unbiased. The team should ask questions about the improvement they want to make, and this way, they will understand the reasons behind any data collection they make.

The second pattern is "**Choose the Means**". This pattern aims to define the data sources from where the data will be collected and the appropriate means to collect it. Data can be collected with tools, techniques, and other approaches to extract data useful in future decisions related to the issue in mind. Decisions made quickly in development might not reflect the reality of the software system. Finding the correct means to collect data is one step further in making better decisions.

The next pattern is "**Software Analytics Planning**". Tasks related to improving code quality might take a lot of time and effort, so they should be done continuously. Decisions made based on real data can avoid technical debt and optimize other future tasks. This pattern is related to planning software analytics tasks and the project by adding them to the backlog and prioritizing it with other regular tasks.

The fourth pattern is "**Analytics in Small Steps**". Like we said in the previous pattern, software analytics tasks can take a lot of time and effort, and the team should understand how to implement them without negatively impact the team. Software teams, especially in start-ups, don't have much time to execute software analytics tasks. The software teams must then distribute tasks related to software analytics through the project iteration, adding information for the team in small portions and prioritize the tasks that give the most critical information and answers to the most vital issues. This way, the team can introduce software analytics related activities without interfering with the regular tasks and priorities.

The fifth pattern is "**Reachable Improvement Goals**". If a software team takes on the challenge of trying to fix every issue in the code, they might never move forward. Without a clear goal, it can be a waste of time for the software team. The team should define reachable goals from the software analytics findings and break down activities to fit with other tasks to reach that goal. By following this step, the team establishes a culture of continuous improvement.

The sixth pattern is "**Learning From Experiments**". The software team might not succeed every time. The team must keep trying to find alternative solutions to a problem, for example, running several data collection experiments for comparing with the current solution or even trying other solutions to improve a goal to compare with the current version.









The seventh pattern is "**Define Quality Standards**". This pattern is about defining thresholds that the team should strive for that they consider valid for a quality issue. It should be done for every quality aspect that the software team decides to monitor.

The last pattern is "**Suspend Measurement**". This refers to suspending the measurement of issues that might take too much effort or cost, issues that might be blocked by other project constraints, and issues that have stabilized and don't have much of a possibility of recurring. This helps to keep the measurements to only the necessary and useless measurements should be discarded.

The canvas is composed of seven blocks. The first is the **key issues**, and it's the first step. It's where the team raises issues that need to be analyzed and improved. The second is **data sources**, and it's where the team identifies what kind of data they need and where to get it to find more about the issue. The **data gathering** comes after, and it's where the teams decide which metrics and tools will be used. The **insights** block is where the team can analyze the results obtained and possible solutions for the issue. The **quality thresholds** phase is where the team defines when the issue is considered resolved and if the issue should be monitored. The **analytics implementation** block is where the team can plan their analytics tasks and prioritize them with other development tasks. The **incremental goals** block is where the team defines reachable goals to start executing their solutions.

From this artifact, two studies were conducted to evaluate the canvas [5]. The first study was to understand the perceived usefulness and ease-of-use of the Software Analytics Canvas and was conducted with six subjects from a course of Agile Projects. The subjects were separated into three pairs. During the study, the teams were asked to plan and manage a software analytics project based on a real case using the Software Analytics Canvas. In the end, the participants were asked to answer a questionnaire. Most participants agreed that the canvas was useful, but some disagreed on the ease of use and found that it was not easy to learn. Some found it hard to understand what to do at certain times.

The second study was a participatory redesign with the same participants in the first study to identify how to improve the Software Analytics Canvas. The participants were asked to give suggestions on improvements and were asked to sketch a redesign proposal. In the end, a discussion took place, and the participants sketched a new version of the canvas. The quality threshold block was subdivided into two parts with a minimum acceptable value and goal to achieve. The bottom of the canvas now has an analytics tasks block with four components: to do, in progress, done, and impediments. Version 2.0 was then created based on this study's results, as we can see in Figure 3.3.

SOFTWARE ANALYTICS CANVAS					
PROJECT: PROJECT NAME				VERSION: 2.0	
 KEY ISSUES	 MEASUREMENTS	 METHODS AND TOOLS	 HIGHLIGHTS	 INSIGHTS	 GOALS
					 DECISIONS
 ANALYTICS TASKS					
To Do	IN PROGRESS	DONE	IMPEDIMENT		

Produced by Choma et al. (2020)

Figure 3.3: Software Analytics Canvas Version 2.0 [5]

3.3.3 Discussion

These approaches give the team the ability to focus on not only assessing the quality of their systems but also improving it. Approaches like the Software Analytics Canvas emphasize the

importance of improving in small steps without harming the team's performance. It is even more critical for smaller companies, like start-ups, that can't take some time off development to focus on improving code quality.

Approaches like the Software Analytics Canvas and PDCA give a framework for teams to continuously improve their code by planning, implementing, and analyzing their findings to find solutions for the problems or even prevent future problems.

There's even more of a lack of approaches to change agile teams' habits towards data-driven decision making [5]. This is where the Software Analytics Canvas approach differentiates from the rest. It is an approach focused in an agile development context for implementing software analytics in software projects. The canvas serves as a backlog of information and tasks where the teams decide every sprint which task should be prioritized and executed. The PDCA promotes continuous improvement of code but still has phases that need to be strictly followed.

Chapter 4

Problem statement and Proposed Solution

In this chapter, we describe this dissertation's scope with greater detail within the context of assessing code quality using continuous improvement methods and software analytics in a start-up environment. Then we introduce the research questions that we want to answer with this dissertation and the methodology established to answer them. Finally, we present the work plan defined.

4.1 Scope

Low-quality code often makes it difficult for the software teams to maintain and develop new product features. Low code quality is often associated with the lack of assessment of the code quality and the lack of understanding of the metrics implemented. Thus, by poorly assessing the code quality or not assessing, the software teams can't assess technical debt when making significant changes and producing value.

Software Quality models give us multiple definitions of what quality should be in software. Some models give us abstract quality aspects and others more specific metrics that should be monitored.

Although these models help us understand what software quality is, this isn't enough for a software team to assess code quality. Software teams need approaches to operationalize these quality models and methods that can provide us a way to evaluate code quality in a structured process. Some of the existing approaches focus more on finding the specific metrics that should be used. Others outline the activities that a team should perform to assess the quality of a software product.

However, assessment approaches don't explain how to act with the data and information gathered. Continuous improvement approaches have a structured process to assess the code quality and find metrics and have steps to act with this information. In smaller companies like start-ups, the development team needs to continuously improve their projects' code quality to manage technical debt intelligently and maintain efficiency and agility in development. Technical debt can be

created but only if done consciously and re-paid in the near future. They have to develop the product while still keeping the product's quality at the right level. There is, therefore, a need for continuous improvement methods that integrate with other software activities.

4.2 Research Questions

Software teams need a software analytics approach to understand the problems existent in their projects, measure these problems with metrics, and track improvement over time. We aim to tackle this problem using the Software Analytics Canvas approach to plan software analytics related activities of software teams in a start-up environment. We want to perform an industrial case study since there is still a lack of studies to validate the approach. Given that, this dissertation addresses the following research questions:

- **RQ1** *"What are the users' perceptions about the usefulness and ease-of-use of the Software Analytics Canvas?"* We aim to evaluate if the approach is easy to use and if the team sees the artifact as useful to improve the code quality.
- **RQ2** *"Are the users more aware of the quality issues of their project and do they understand the reason behind the metrics being monitored?"* We aim to evaluate if the approach can help the users the implemented metrics by understanding the problem first. By users, we mean not only developers but also product owners and managers.
- **RQ3** *"How does the Software Analytics Canvas affect the software system's quality over time?"* We aim to evaluate how the approach will help the software teams improve their code quality over time. We want to see the evolution of the canvas content and the information obtained throughout the study.

4.3 Methodology

In this dissertation, we aim to understand the benefits of using a software analytics approach for continuously improving software quality, like the Software Analytics Canvas, in a start-up environment. For that reason, we start by studying the related work in the area to understand what has been achieved in the software quality improvement field. In this review, we compare multiple software quality models, software quality assessment approaches, and continuous improvement approaches, which help us understand better the problem we're trying to solve. The problem is defined from this analysis, and research questions that we want to answer with this work are established.

To answer the research questions, we will conduct an industrial case study composed of three phases with two software teams. The research method used in this case study will be Cooperative Method Development [8] which can be described as a domain-specific adaptation of Action Research [1], this domain being software development. With this method, researchers take a

clear stance in participating with the software engineers in the practices they are observing. This method has three main phases derived from Action Research. The *first* phase is understanding the practices that currently exist and what the existing problems are. The *second* phase is deliberating measures that will improve the current situation. The *third* and last phase consists of implementing the measures and observing the improvements in which the researchers participate as participant observers.

In the first phase, we will analyze the participating teams. We aim to understand the current software analytics practices to really understand the problems they currently have.

The second phase will be put into practice by getting the teams familiarized with the approach. This will be achieved through a demonstration, in which we will present the canvas to the participant teams by explaining all the approach's lanes and giving concrete usage examples. After the demonstration, a survey will be handed to the participants to assess their initial understanding and perceived utility of the approach.

In the third phase, the participant software teams use the Software Analytics Canvas to plan their software analytics activities and make decisions based on the obtained insights in an iterative, systematic, and continuous manner. This phase will last a total of ten weeks.

The participant teams have four members each, and they work independently from each other as the company's product is divided into several independent micro-services, which facilitates this arrangement. One of the teams focuses on front-end and back-end development, and the other only on back-end development.

Both teams adopt the Scrum methodology. They have a planning meeting before the start of every sprint, daily meetings where they discuss current problems and progress in the sprint board, and sprint review and retrospective at the end of the sprint.

The Software Analytics Canvas would be assessed in the sprint planning to determine the tasks to be done during the sprint. Daily meetings would be used to communicate the results and progress of the software analytics tasks. And the review meeting should be used to discuss the software analytics sprint findings to make decisions based on the data found. The sprint planning session uses these findings and conclusions to set the tasks for the next sprint.

Possible small changes to the canvas can be made during the review meetings if some lane isn't clear. However, the teams will be using the canvas as is, in the beginning, to form opinions and fully understand the approach as it was defined.

During the period of the case study, the participants answer a survey after every sprint. These surveys help us understand the perceived usefulness and ease-of-use of the canvas, if the approach allows the team to be more aware of the quality issues and understand the metrics implemented, and if the canvas helped improve the code's quality. Other measurements will also be taken along this phase. We will analyze the time spent on the canvas in every meeting to understand the percentage of the meeting dedicated to the approach. During the sprint itself, the percentage of points spent on software analytics tasks will also help understand how much work is done in the sprint dedicated to software analytics compared to development tasks. Finally, the evolution of the canvas will also be recorded by taking screenshots at the beginning and end of every sprint. This

will help understand the evolution of the canvas and the evolution of the information gathered on the source code.

After the development phase, the study participants participate in a session to improve the approach. The participants voice their opinions on the canvas and express which refinements should be made to the canvas.

Chapter 5

Industrial Case Study

In this chapter, we describe the case study in detail starting with the goals of the study. We then explain the design of the study and we provide answers to the research questions with a in-depth result analysis. We also talk about the start-up environment and how it impacted the study and present a new pattern detected with this study. We end the chapter by referencing some possible threats to the validity of the study and a discussion of the chapter.

5.1 Goals

The goal of this case study is to validate the Software Analytics Canvas approach by answering the 3 research questions defined in section 4.2. The 3 research questions compose 3 main topics that we want to validate.

The first one is the **utility and ease-of-use** of the approach. Since the Canvas is an approach and a tool, it is important that the users who are using it feel that they are using something useful that produces a big enough value so that it is worth it to keep using it in the future. We want to answer this by understanding a few metrics. We want to understand how the participants perceived the utility and ease-of-use of every lane of the Canvas. In the last survey we also question the users about the learning experience in order to better understand if the learning curve is steep or acceptable.

The second one is the **awareness and understanding** of the issues and metrics in the system over time. The Software Analytics approach aims to make the software quality issues accessible and understandable to all the levels of a company and raise awareness to the current state of the system. We assess this with topics over time in the surveys to see how they evolve with the time using the approach.

The third one is the **evolution of code quality of the system** over time. We want to assess if the teams that used the Canvas could turn the issues and results actionable in order to improve the quality of the system. This was done mainly by monitoring the chosen metrics related to the issues defined in the Canvas.

5.2 Study Design

This section of the document is dedicated to introducing the general design of the case study. We will be describing the environment and the participants, the structure, but also, how the data is collected and analyzed.

The **participants** of this case study are all from the same company. There is two main groups of participants: the developers and the managers. From the developers group, two teams participated in this experiment. One of the teams has four developers mainly focused on back-end development. The other team had 5 developers, three of them focused on back-end development and two of them focused on front-end development. These two teams worked with the Scrum methodology on a daily basis. This group of participants were the ones mainly using the Software Analytics Canvas on a daily basis. The managers group is composed of product managers and the Chief Technology Officer (CTO) of the company. This group was not interacting directly with the Canvas but was indirectly impacted by participating in planning and retrospective sessions where the process is discussed.

The **protocol** of the study can be classified in three main phases: the introduction to the process, the usage of the process and the study closure.

The **introduction phase** was the first two weeks of the study. At first, the development teams were presented with a survey to assess their software analytics experience. The teams and product managers were then introduced to the process with a small presentation in order to get to know how the Canvas is used. And in the end, right before the first sprint started, a first session with the Canvas occurred with both teams separately where they populated the Canvas with issues they had in mind from the time they spent working on the company system.

The **sprints phase** started with the start of the next sprint and lasted a total of 5 sprints of development. Both teams used their own version of the Canvas, where they registered issues, metrics, result analysis, goals and decisions specific to their own team. All the tasks that came from the Canvas iterations were added to their own sprint boards. At the end of each sprint, the team members were presented with a survey in order to gather some metrics over time related to ease of use, utility and improvement of the system's quality.

The **study closure phase** of the study was composed of one last session with each team where they discussed their experience with the process, what they liked and what they would change about it. After these sessions, a survey was handed to all the team members to assess the difference between the Canvas and their old software analytics related activities, the learning experience and the positive and negative consequences of using the Canvas. Another survey was handed to the product management teams and the CTO to assess their willingness to be more involved with the process and if they consider that the Canvas produces actionable information for them.

There are 3 main **data sources** for this study. The first one is the **surveys** answered in the beginning, at the end of each sprint and at the end of the experiment. The second data source is **Jira Software** where we can gather sprint metrics. The third one is **Miro** where we simulated the Software Analytics Canvas and gathered metrics from it. The fourth and last source is **Software**

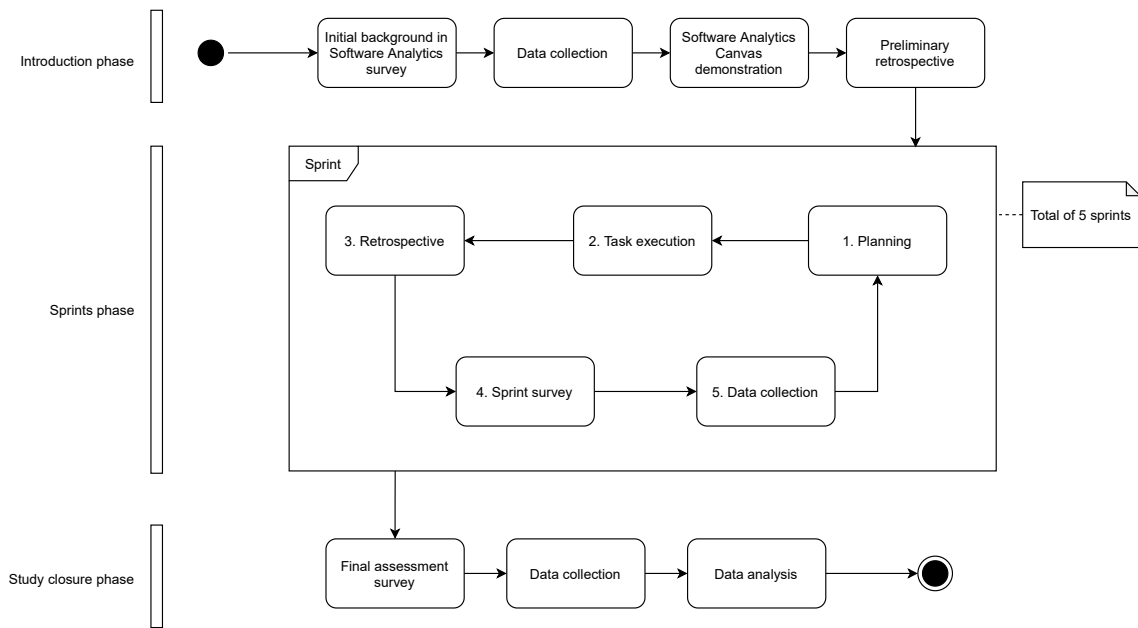


Figure 5.1: Study Design Diagram

Analytics Tools and scripts, where we gathered information related to code quality metrics and their improvement.

Regarding **data collection**, a great majority of the data collected was from the Google Forms surveys. These surveys were handed to the participants various times, including the start of the experiment, at the end of every sprint and at the end of the experiment. For the surveys, we used Likert items with the values between 1 and 5 where each number represents the following interpretations: 1) Strongly disagree, 2) Disagree, 3) Neutral, 4) Agree and 5) Strongly agree.

At the end of every sprint we also had a phase of data collection where we gathered several metrics besides the surveys. In this data gathering we collected from Jira metrics related to the tasks completed and not completed during the sprint and the percentage of software analytics tasks in that sprint. We also gathered timings of the retrospective meetings to see the time added in meetings in every sprint. And finally, we also gathered information from Miro related to the usage of the Canvas like the amount of cards added to every lane in that retrospective meeting.

At the end of the experiment, we gathered extra information related to the improvement of metrics. This was mainly done by checking the documentation of every improvement. The data gathered for these documents were mainly from software analytics tools in Gitlab, SonarQube and manual scripts.

The **data analysis** was mainly performed with spreadsheets. All the data collected, including the survey data, sprint metrics, Canvas metrics and code quality metrics were aggregated in multiple spreadsheets. From the data aggregated in these spreadsheets we could calculate means, standard deviations, percentages and generate plots to go with the analysis.

5.3 Results Analysis

In this section we analyze the data gathered from the four months of research in a start-up. We assess the company's background in software analytics, the process of using the Canvas in two different teams, the perceived usefulness, the improvement of quality awareness and the evolution of the system's overall quality.

5.3.1 Company Background in Software Analytics

The company's background in software analytics is analyzed through a survey in the introduction phase of the experiment. The survey was composed of questions regarding their awareness of the current state of the system's quality, the state of software analytics in the company and how they use software analytics. This survey was done before any presentation of the Software Analytics Canvas to avoid any biased responses.

From this survey, as represented in Table 5.1, we find that the perception of the participants was that they were fairly aware of the system quality. However, they had not established many software analytics practices, did not consider that they were using appropriate software tools, didn't really make much use of code quality metrics and did not discuss about software quality outside code reviews. The only code quality metric they really used was code coverage to reduce the number of bugs and speed of development.

Table 5.1: Responses to the background survey

Questions	\bar{x}		σ
Q1	3.7		0.8
Q2	2.3		1.0
Q3	2.4		0.8
Q4	2.9		0.9
Q5	3.0		1.4
Q6	1.9		1.1
Q7	3.0		0.8
Q8	3.6		0.5
Q9	3.2		0.8
Q10	3.2		0.8

Q1: I am aware of the current state of quality of our software system's code base.

Q2: My team has established Software Analytics practices.

Q3: My team is using appropriate Software Analytics tools.

Q4: My team currently makes use of code quality metrics.

Q5: The implemented metrics are used in order to solve an existing problem or reach an objective.

Q6: My team frequently includes Software Analytics tasks in the sprint planning.

Q7: My team dedicates some time of our sprint to refactoring activities.

Q8: I refactor code during a sprint task in order to deliver better code.

Q9: My team creates refactor tasks in order to improve the code quality incrementally.

Q10: My team assesses the impact of these refactors.

They also didn't really plan for any code quality refactoring tasks in the sprint planning as the sprint backlog was always decided by the product team and not the development teams. The product teams always decides which user stories and tasks are to be done in each sprint. Most of their code quality improvements were made during normal sprint tasks. We can assume that this awareness of the system quality was mostly subjective to the experience of the participant in the company's system. And, as we will observe in Section 5.3.4, in the final survey, the participants admitted that they were not as aware as they thought regarding the state of quality of their system.

5.3.2 Usage of The Software Analytics Canvas

The participant software teams used the Software Analytics Canvas during five sprints of development. During these five sprints, the teams iterated through the Canvas to register new issues, register findings and define goals and solutions.

Both teams used a different variation of the Canvas from the one explained in Section 3.3.2. This was due to the fact that the company was working remotely and they already had a software for task management which was Jira Software where they managed the sprint tasks. Jira is also made for project management and gives the teams several useful metrics to improve their software development processes. The core of the Canvas remained the same but was separated in two different boards, therefore, calling it Version 2.1 instead of 3.0. The first board, presented in Figure 5.2 was composed of the top part of the Canvas, which includes the following lanes: Key Issues, Measurements, Methods and Tools, Highlights, Insights, Goals and Decisions. This board was created on Miro, a visual collaboration platform.

The second board, presented in Figure 5.3, was composed of the bottom part of the Canvas, where the tasks are registered. This part of the Canvas was implemented as a normal Kanban board dedicated to general technology tasks and was created on Jira Software that the teams already used for project management. On this board, the teams registered all the tasks generated from the Canvas and then connected these issues to their own boards so that they could manage the progress during the sprint.

Regarding the lane usage, as we can see in Figure 5.4, the preliminary retrospective had a big contribution in the *key issues*, *measurements*, and *methods and tools* section. This is due to the kick-off meeting where each team discussed the main issues they currently felt were critical. Another event that caused this initial boost was the meeting before the usage of the approach where we had a presentation to explain the Canvas and we encouraged them to think about possible issues.

After the large contribution in issues from the kick-off sprint, the rest of the sprints were mainly focused in *highlights*, *insights*, *goals* and *decisions* and the first three lanes (*Key Issues*, *Measurement*, and *Methods and Tools*) were less used due to the amount of work they had to do to investigate the issues already defined. The only sprint that was an exception was the fourth sprint, where the team was late on the sprint objective and the tasks that had to be left behind were the software analytics ones. Although no tasks were completed, a key issue was found and added to the board in that sprint.



Figure 5.2: Software Analytics Canvas Version 2.1 Issue Section

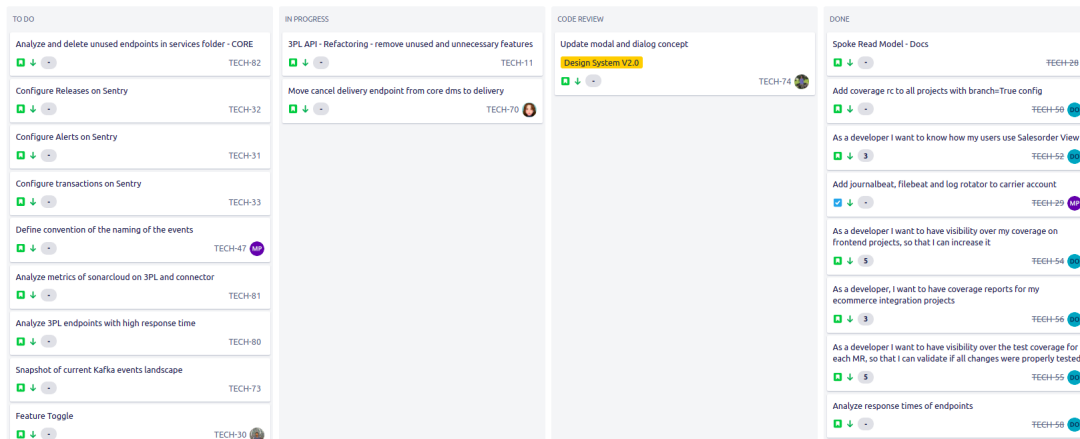


Figure 5.3: Software Analytics Canvas Version 2.1 Tasks Section

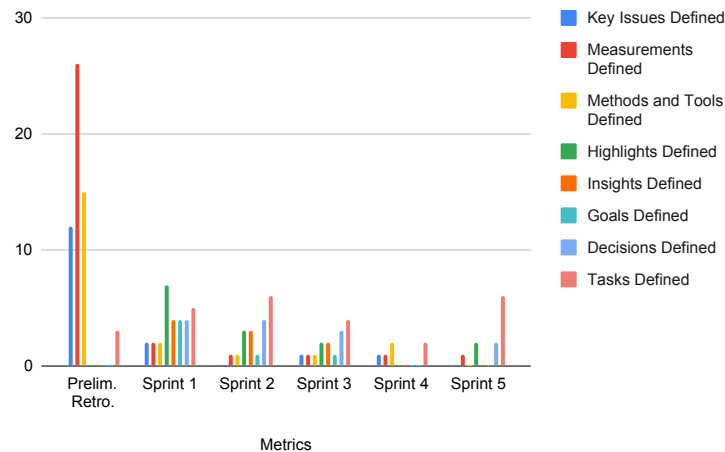


Figure 5.4: Software Analytics Canvas lane usage per sprint

And, as we can observe in Table 5.2, in average, the software analytics tasks would be around 12.9% of the total planned tasks, which compared to before the Canvas, it's a great improvement as no software analytics tasks were being added to the sprint. As they were almost always completed, software analytics tasks would compose of around 18.1% of completed tasks in a sprint. As the approach was meant to be used, some of the tasks generated from the Canvas would be chosen to do in each sprint, occupying a small percentage of the team's efforts. This way, the team could maintain its velocity while improving the system's quality over time. The tasks were chosen based on the priority of the issue and the results found on previous sprints.

Table 5.2: Software Analytics tasks to Regular Tasks comparison by sprint

Metrics	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Total
User Stories Planned	26	22	18	22	20	108
SA Tasks Planned	3	5	3	3	2	16
User Stories Completed	16	11	9	10	13	59
SA Tasks Completed	3	5	3	1	1	13
Total User Stories	29	27	21	25	22	124
SA User Story %	10.3%	18.5%	14.3%	12.0%	9.1%	12.9%
Total User Stories Completed	19	16	12	11	14	72
SA User Story Completed %	15.8%	31.3%	25.0%	9.1%	7.1%	18.1%

Regarding the meetings, from what we can observe in Table 5.3, the teams had one recurring meeting at the end of every sprint where they looked at the Canvas and the results of that sprint. The software analytics retrospective time was in average, half the time of the total retrospective time, and since the sprints were of two weeks, it took almost no time of their work for the value it brought.

Almost all the *Key Issues* were defined in the beginning of the experiment, hence the big input

Table 5.3: Software Analytics Retrospectives to Regular Retrospectives comparison by sprint

Metrics	Prelim. Retro.	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Total
Normal Retrospective (min)	90	125	84	128	122	128	677
SAC Retrospective (min)	210	165	112	96	138	45	766
Total Time (min)	300	290	196	224	260	173	1443
SA Time %	70.0%	56.9%	57.1%	42.9%	53.1%	26.0%	53.1%

from the preliminary retrospective that we can observe in Figure 5.4. The rest of the experiment was mostly focused on evaluating the metrics defined for these issues. By what we can see from the comparison between the software analytics tasks and normal user stories is that only a small portion of the team's time was used for software analytics. In the final survey handed to the managers, they also thought that they didn't feel any slow down from the team output compared to before the Canvas.

5.3.3 Usefulness and Ease-of-use Perception
















To evaluate the usefulness and ease-of-use of the approach, we created a survey that team members answered at the end of every sprint. We sent the same questionnaire after every sprint with questions presented in a *Likert* scale where the answers are a number between one and five. The lowest value corresponds to "strongly disagree," and the highest value corresponds to "strongly agree."

As shown in Table 5.4, this survey contained questions to assess the understanding and ease-of-use of the canvas. Besides the general question about ease-of-use, we asked for every lane of the Canvas if it was easy to use and assessed the confidence in their understanding of the lane. As we can see from this table, the averages of understanding are superior to the averages of ease-of-use. We can interpret that the Canvas is easier to understand than it is easy to use. This is consistent with the observation done during the sessions in which the Software Analytics Canvas was used. The teams understood the concepts well, but still, we could notice difficulty when adding items to the Canvas.

Nonetheless, the values for the **understanding** are relatively high, which means that the approach is quite easy to understand. Although the values of ease-of-use are lower than the understanding values, they are still in the range of three to four, which is between neutral and easy to understand. Nonetheless, it is still a positive and high result.

The understanding had a slight variation over time. As we can see in Figure 5.5, it had the largest positive variation from the first sprint to the second sprint. We can interpret here that, as it is expected, the first use of a new approach can be quite overwhelming and can be hard to understand in the first use, which explains the growth in the second sprint. The curve then declines in the third sprint, which can be justified because one of the teams didn't have relevant results in that sprint which can explain the sentiment decline. The fourth sprint stayed the same, which can also be explained by the fact that both teams did not have time to implement the tasks related to

Table 5.4: Average of responses to the sprint survey regarding understanding and ease-of-use of the Software Analytics Canvas

Questions	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Mean	
Q1	4.0	4.0	4.2	4.0	4.3	4.1	
Q2	4.1	4.6	4.3	4.4	4.6	4.4	
Q3	3.9	3.9	4.3	3.8	4.0	4.0	
Q4	4.1	4.6	4.3	4.2	4.3	4.3	
Q5	3.7	3.6	4.2	3.8	3.9	3.8	
Q6	4.3	4.4	4.3	4.4	4.4	4.4	
Q7	3.6	3.9	4.0	3.4	4.0	3.8	
Q8	3.9	4.3	4.2	4.2	4.3	4.2	
Q9	3.6	3.3	4.0	3.8	3.9	3.7	
Q10	3.7	4.1	4.2	4.2	4.4	4.1	
Q11	3.6	3.6	4.0	3.8	3.9	3.8	
Q12	4.3	4.4	4.3	4.2	4.3	4.3	
Q13	4.0	3.9	4.2	4.2	3.7	4.0	
Q14	4.1	4.4	4.3	4.2	4.4	4.3	
Q15	3.9	3.9	3.8	4.2	4.0	4.0	

Q1: I found it easy to use the Software Analytics Canvas in this sprint.

Q2: At this point, I'm confident in my understanding of the concept of Key Issues.

Q3: In this sprint, I found it easy to define Key Issues on the Canvas.

Q4: At this point, I'm confident in my understanding of the concept of Measurements.

Q5: In this sprint, I found it easy to define Measurements on the Canvas.

Q6: At this point, I'm confident in my understanding of the concept of Methods and Tools.

Q7: In this sprint, I found it easy to define Methods and Tools on the Canvas.

Q8: At this point, I'm confident in my understanding of the concept of Highlights.

Q9: In this sprint, I found it easy to define Highlights on the Canvas.

Q10: At this point, I'm confident in my understanding of the concept of Insights.

Q11: In this sprint, I found it easy to define Insights on the Canvas.

Q12: At this point, I'm confident in my understanding of the concept of Goals.

Q13: In this sprint, I found it easy to define Goals on the Canvas.

Q14: At this point, I'm confident in my understanding of the concept of Decisions.

Q15: In this sprint, I found it easy to define Decisions on the Canvas.

Software Analytics that they decided to make in that sprint. The fifth sprint had a better result which could be the cause of the increase in that sprint.

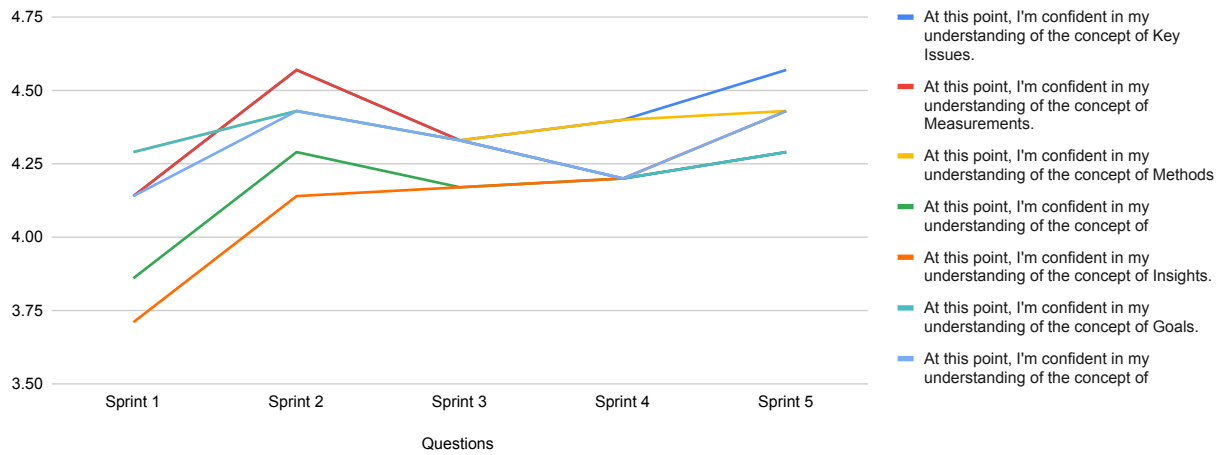


Figure 5.5: Software Analytics Canvas Lane Understanding per Sprint

As for the **ease-of-use**, we can see in Figure 5.6 that these variations are connected to the results of the tasks of each sprint. Some results are easy to register, and others not as much.

The decline from the first sprint to the second can be justified because the findings did not bring much value. One team had good results but was related to the other team, so nothing of importance was registered in the Canvas. The third sprint had a significant increase, which can also be directly associated with the results. In this sprint, important results made it easy to register results in the Canvas and a tool setup that proved quite valuable to the teams. There was also a meeting between the teams to explain the finding from the previous sprint. We can see a decline in the fourth sprint, which is also related to the fact that none of the teams had time to complete all their software analytics tasks.

We also added some questions to the sprint survey to obtain some answers on the topic of usefulness. These questions are presented in Table 5.5. The usefulness results of the surveys were fairly high from the start, and we can interpret that this is because the teams were not previously using any software analytics practices, as shown in Section 5.3.1. From the values of Question 2 in Table 5.5, we can see that the team never considered the Software Analytics Canvas as a process that took a lot of time of the sprint. As we can observe in Table 5.2 from Section 5.3.2, the software analytics tasks never surpassed the 20% mark of the total tasks planned for a specific sprint.

Similar to the other metrics, the **usefulness** metrics also showed slight variations. In general, both questions one and three had high values, as we can see in Table 5.5, which means that the teams regarded that the process brought value to them and the time spent was well spent. Except for Sprint 4 where these metrics dipped, as we can see in Figure 5.7. As mentioned before, the

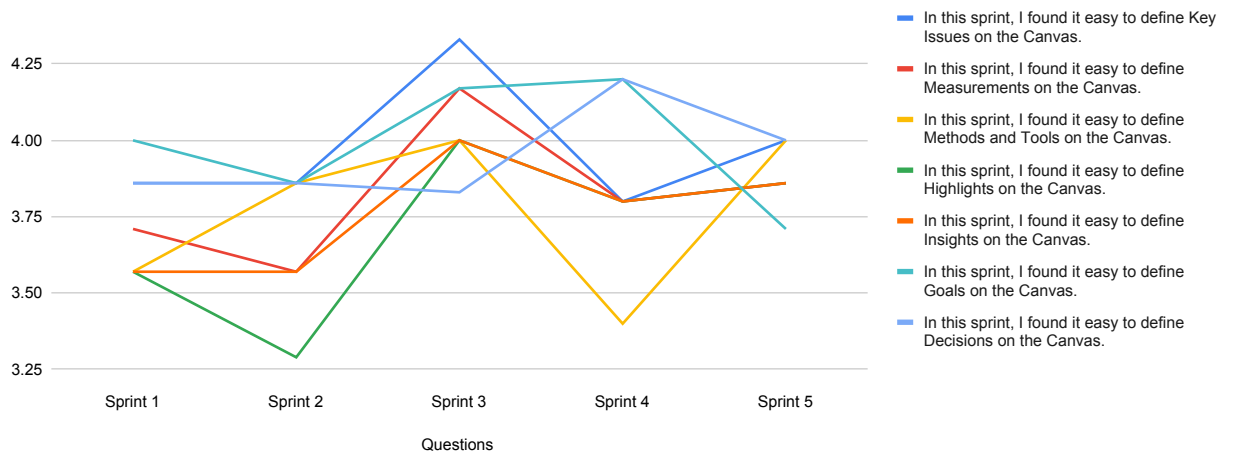


Figure 5.6: Software Analytics Canvas Lane Ease-of-use per Sprint

Table 5.5: Average of responses to the sprint survey regarding the usefulness of the approach

Questions	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Mean	
Q1	4.3	4.1	4.0	3.8	4.3	4.1	<div style="width: 80%; background-color: #808080;"></div>
Q2	2.4	2.3	2.5	2.4	2.0	2.3	<div style="width: 40%; background-color: #808080;"></div>
Q3	4.6	4.6	4.5	4.4	4.6	4.5	<div style="width: 90%; background-color: #808080;"></div>

Q1: The Software Analytics Canvas was useful in this sprint.

Q2: The Software Analytics process took a considerable time off this sprint.

Q3: The time spent on Software Analytics in this sprint is time well spent and beneficial.

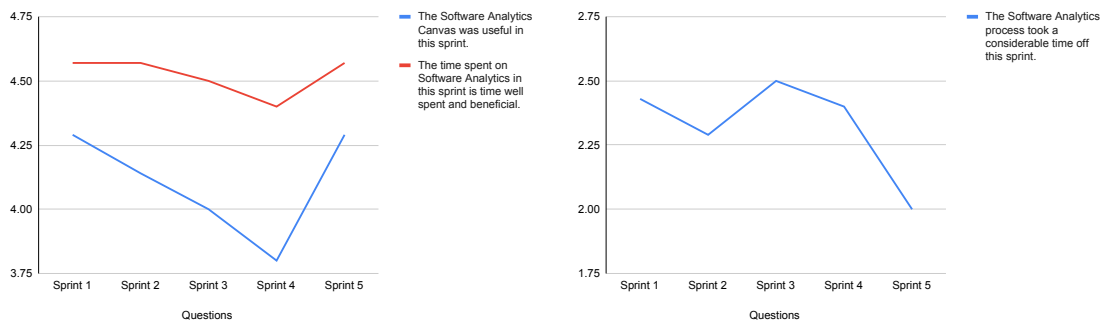


Figure 5.7: Software Analytics Canvas Usefulness Perception per Sprint

tasks of Sprint 4 were not completed, so it made the meetings and time spent less useful to generate new cards on the Canvas.

We also assessed the product managers and CTO regarding the approach. As we can see in Table 5.6, they felt like the Canvas was easy to understand and that it was useful. We presented them with some of the results and asked if what the teams found was relevant to their work which got a positive response. They also agreed that a process like this is important and dedicating a percentage of the team's effort is essential. A majority of the managers also wanted to be more present in software analytics sessions. This would help them be more aware of the issues and improve their decision making when it comes to deciding what should be done in the next sprint.












5.3.4 Code Quality Awareness

At the beginning of this experiment, a kick-off survey was handed to the participants as describe in Section 5.3.1. In this survey, three questions are of importance to this topic. As we can observe in Table 5.7, the participants felt reasonably aware of the system's quality. Yet, as we can see by the answers to questions two and three, the teams did not adopt many software analytics practices and were not using code quality metrics. Therefore, we can conclude that their awareness was based on their intuition of working in the system for a while. While this indeed grants you some degree of understanding of the state of quality, many problems can go unnoticed.

At the end of each sprint, with the sprint survey, the participants were also asked if they felt more aware of the state of quality by the end of the sprint. As we can see through the values of each sprint on Table 5.8, on each sprint, the participants felt more aware of the state of quality by simply measuring new metrics every sprint as the general mean of every sprint was 4.02, it is a positive result.

After the experiment ended, the participants were surveyed one last time. In this survey, there were some questions of relevance to the participants' awareness regarding the state of quality of the system. As we can see in Table 5.9, the participants affirmed that they felt that their system had technical debt and that at the time the study began they were not paying it the required attention. Some said in an open question that this lack of awareness was mainly related to fast growth,

Table 5.6: Responses to the final survey to the managers regarding usefulness, understanding and ease-of-use

Questions	\bar{x}		σ
Q1	4.3		0.5
Q2	4.3		0.5
Q3	3.8		1.0
Q4	4.8		0.5
Q5	4.8		0.5
Q6	4.3		1.0
Q7	4.3		0.5
Q8	4.8		0.5
Q9	4.5		0.6
Q10	3.8		1.0

Q1: I feel that the concept of the Canvas is easy to understand.

Q2: I feel that the concept of the Canvas is useful.

Q3: 70% of the the Core and 90% of the 3PL service endpoints were not being used at all. This is relevant for my work.

Q4: Long waiting times can cause a bad experience for the user. Several endpoints take more than 1 second to complete and even some take more than one minute to complete. This is relevant for my work.

Q5: Code not covered by tests can be cause of bugs in the system. Some projects had between 50% and 70% of code covered by tests. This is relevant for my work.

Q6: These types of findings are important to decision-making when prioritizing tasks for the next sprint.





Q7: I feel like the software analytics process is important and that we should keep using it.

Q8: I feel like dedicating a small percentage of the team tasks to software analytics is beneficial.

Q9: I feel like in my role in the company, understanding the quality of the system is relevant to the decisions I take.

Q10: I feel like participating in the software analytics sessions would be beneficial.

Table 5.7: Responses to the kick-off survey regarding the code quality awareness


Questions	\bar{x}		σ
Q1	3.7		0.8
Q2	2.3		1.0
Q3	2.9		0.9

Q1: I am aware of the current state of quality of our software system's code base.

Q2: My team has established Software Analytics practices.

Q3: My team currently makes use of code quality metrics.



Table 5.8: Average of responses to the sprint surveys regarding the code quality awareness

Questions	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Mean	
Q1	4.1	4.4	3.8	4.0	3.7	4.0	

Q1: Following this sprint, I feel that I am more aware of the issues that the company projects have.

focusing on new features, pressure to deliver quicker, and few team members. These are some of the main characteristics of software teams in Startups. There is an urgent need to provide new features fast, and sometimes, code quality is compromised in favor of time to market.

Table 5.9: Responses to the final survey regarding the code quality awareness before the Canvas







Questions	\bar{x}		σ
Q1	4.4		0.5
Q2	2.3		0.8

Q1: We had technical debt in our system.

Q2: We were paying the required attention to the system code quality.

In this final survey, we also evaluated the impact of the Canvas on the team's awareness. As we can see by the results in Table 5.10, the participants felt more aware than before by using the Canvas on every sprint. It made the participants use more appropriate tools and making more use of code quality metrics. They also agreed that putting the issue first instead of the metrics made them more aware of why they used a specific metric. They also felt that the Canvas helped the teams encounter severe issues and some that could have been left unnoticed if they didn't use Software Analytics. We can see that the awareness has been a positive consequence on the process.

Table 5.10: Responses to the final survey regarding the impact on code quality awareness

Questions	\bar{x}		σ
Q1	4.1		0.7
Q2	4.0		0.6
Q3	4.4		0.8
Q4	4.6		0.5
Q5	3.9		0.9
Q6	4.4		0.5

Q1: I am more aware of the current state of quality of our software system's codebase.

Q2: We are using more appropriate Software Analytics tools.

Q3: We are making more use of code quality metrics.

Q4: The Software Analytics Canvas process helped the team understand the metrics chosen by first identifying the issues.

Q5: The Canvas helped the team encounter serious issues.

Q6: Without the Canvas, some of the issues could have been left unnoticed.

5.3.5 Code Quality Improvement

The overall system quality is a complex metric to evaluate, and we can't tell how it has evolved since the start of the Canvas usage. But, we can determine the improvement of the issues that were defined in this process. From the issues defined on the boards, we can name a few that had a remarkable evolution.

The first significant metric defined was **code coverage** which was defined for both teams. For one team, the issue that started this process was the quantity of bugs they had in the system. For the other, it was associated with the perceived risk that new developments could break existing features. Both teams sought to improve the way they measured the code coverage. The more recent services had very high code coverage, but they noticed that older and legacy projects had low code coverage despite that new code implemented in these projects was always tested. The teams tried to find which parts of the system were lowering this metric and found out that most of the code that didn't have tests was not being used. This generated a new issue which was the amount of dead code they had in these older projects.

As they noticed that they had a lot of **dead code**, it became one of the main issues. They tried to identify which parts of the system were dead code, but as these services have the main goal to provide APIs, it was challenging to find a value for this metric. Since all the code was "being used" in endpoints, the typical dead code tools didn't work here. Endpoint profiling tools were not an option since it reduces the performance of the API by a lot, and they already had performance issues. The solution was then to find all the endpoints defined in the application, which is easy to obtain with some tools, and compare these endpoints to endpoints present in the Nginx logs stored in the production machines. After this test, we found that around 70% of the service's endpoints were not being used for the application chosen from one team. For the service selected by the second team, around 90% of the endpoints were not being used at all. In this case, both teams took a different approach to solve this issue. The first team decided to clean the whole application in one sprint. Most of the endpoints and associated code were deleted, which reduced the complexity of the application and increased the code coverage mentioned above by 20%. The second team took a more passive approach. They divided the application into several sectors and planned the clean-up for several sprints. Two sectors have already been cleaned up, and they will improve in the following sprints. The second team adopted this approach since the application they are responsible for is a lot more complex and more critical than the other team's. Since the task is large and daunting, a division in small steps was a better option. The team applied the SMALL STEPS FOR ANALYTICS pattern [6].

Another critical issue they had in their system was the system's performance. The main problem was that the **response times** from their APIs were considerably high. The percentile 50 and percentile 95 were analyzed, and some endpoints were surpassing the one-minute mark, and a large portion of endpoints was surpassing the one-second mark in the percentile fifty. The teams organized the response times from highest to lowest and decided that the ones with more than one minute should be taken care of first. As they were still cleaning up the applications, this wasn't the main focus until one of the clients reported spending too much time waiting for a specific action to finish the application. The endpoint responsible for that action was the highest percentile fifty, with a value of 2.13 minutes. The task to fix this issue moved directly to the next sprint planning and was fixed very quickly. The issue was related to an event that was being fired but was not being used at all. From the tests done in the staging environment, the process became 660 times faster than before. The issue took very little time to fix and solved a big issue for the clients.

Every time the process was used, it called the endpoint several times. So an operation with 100 calls to the API endpoint that for the client used to take twenty-two minutes only takes around two seconds with the update. And this is a small example. Some clients reported waiting a whole day for the process to finish.

5.4 Challenges and Possible Improvements

During the usage of the Software Analytics Canvas, the teams faced several problems that made the Canvas harder to use than they expected. For some problems, the teams managed to find quick solutions that helped but not every problem had a solution. In the last weeks, we had a final meeting where the teams discussed some of these problems. We also sent a survey at the end where we let the participants share their opinions anonymously and the teams suggested some possible alterations to the Canvas.

- **Highlights and Insights** — Participants said that separating the two lanes did not bring additional value and made it hard to define the cards for each lane. Although the concepts were easy to understand, in practice, it was hard to differentiate the two lanes. In the end of the research, one of the teams proceeded to join the two columns to one called "Highlights".
- **Tracking cards** — This was discovered in the first iterations of the Canvas. As the Canvas is composed of vertical lanes, there is no apparent flow that connects an issue together, as opposed to a table where an issue and the correspondent cards are maintained in the same line. In the beginning, one of the teams found a way to connect the cards together, which solved the problem. The way they solved the issue is by giving colors to the cards corresponding to a specific issue. It was good solution although they still said this was a issue. Keeping the cards corresponding to an issue in the same line as if it was a table also helped but these solutions were improvised by the teams and are highly dependent on the technology used to represent the Canvas. In this case, we were using Miro boards which provides white-board like solution with cards.
- **Progress** — There is no clear way to see how the issue is progressing. The issues are created in the board and stay there while the issue is still being measured. This can create confusion when there are several issues on the board, which makes it even harder to know the progress of each issue. In the last meeting, some of participants made a suggestion to help in this sense. They proposed that the Canvas was implemented in the same way as an Kanban board, where an issue starts in the beginning of the board and goes through several steps until they reach a "Done" lane. They proposed this board as an addition instead of substitution. The Canvas board is useful to keep information and decisions but an additional board brings value in the way that the team knows the state of every issue. The main disadvantage of this solution is that one extra board would be too many boards for the team members to keep track of everything. We could join the boards in only one but the description of

measurements, tools, highlights and insights stay hidden in the description of the card of the issue as compared to the current version of the Canvas. To have the information as visible as it currently is, the cards would be too large and would not be as intuitive to use.

5.5 Threats to Validity

Although we could answer the research questions defined at the beginning of the experiment, there are still some threats to the results. These threats could cause deviations from the results and compromise the validity of the experiment.

- The first threat is **response bias**. Response bias is a term for responses to interviews, surveys, or questionnaires that bias the response from the correct and honest response [10]. In this experiment, we were using the Cooperation Method Development research method [8]. As this method states, researchers participate with the engineers in the practices they are observing. This means that we, as researchers, we're closely working with the participants. As there is proximity to the participants, there might be some response bias to please the researcher. In our case, it is important to mention that some responses could have been lower than they were, making the slight variations in the results essential to interpret. Because of this, in a Likert scale question in a survey, a slight variation from a 4 to a 3 can already mean a lot more than we think.
- The second threat is that the study was done in **only one company** and this company being a **start-up** might make it raises the question if the study can be generalized to other projects. In the world of technology, start-ups are very different than big corporations. There is a lot less organization, the teams are small and multi-disciplinary, and they require fast delivery of new functionalities. This means that code quality is not their primary focus, and an approach like this can bring a lot more value than to other types of companies. The study needs to be performed in more types of companies to understand the approach's utility in different environments.

Chapter 6

A New Observed Pattern: Embedded Improvements

Improvement tasks that emerge from the use of software analytics are often large and difficult to estimate. This type of task can often appear daunting to start and can be left untouched in the backlog. In legacy systems, this type of issue might not even be worth fixing since the codebase is old or the quality has not been the main focus, and most developers might not have worked on that specific part of the system.

6.1 Problem

Software analytics practices often generate technical improvement tasks that imply a considerable effort, and therefore get frequently left behind during planning and may never end up being implemented.

There usually is a **lack of understanding** from the end-users and product owners, which are responsible for the product direction, regarding code-quality issues. They may not be able to identify internal quality concerns and understand the importance of dedicating significant effort to addressing them. Taking on considerably large technical improvements may lead the team to not deliver any value to the project or the company for entire iterations.

There is also a significant **effort** in some tasks derived from software analytics findings, and one can say that simply dividing the issue into smaller tasks will resolve this problem. The team will be able to implement them over time by implementing SMALL STEPS FOR ANALYTICS. Although this might be true, it might not be enough if future developments contribute to the problem itself and, sometimes, the problem might be too complex to be divided into smaller tasks. The task is indeed being resolved, but this doesn't necessarily imply any plans to prevent the issue from happening.

6.2 Solution

Change the development practices to embed improvements gradually and continuously in other tasks.

Changing the development practices is essential for this pattern to work, and there are two complementary ways a team can implement this pattern.

This first way is when a team changes their *DoD* (Definition of Done) to prevent an issue from happening again. The *DoD* is a checklist that needs to be completely done before considering a task as completed. The team will have the responsibility to make sure that the issue is not happening before closing a task. This way, the issue does not need to be brought up during the planning. It will simply be an underlying requirement when doing a specific task. The team must still be aware of older developments which still need to be resolved.

Another way one can implement this pattern is by considering the issues during technology refinements. If the issue exists, it should be addressed when defining a new development or User Story tasks. The solution to a problem the team is trying to solve should already be prepared to avoid making the same mistakes repeatedly. This can be done by defining refactoring tasks before implementing the new development to simplify development while tackling the issue, which facilitates future developments.

6.3 Consequences

There are several consequences when implementing this pattern. On the positive side, one of them is the **guarantee** that the problem won't reproduce. When the team considers the issue on new developments or includes a specific task on the *DoD*, it is a prevention technique to stop a particular issue from happening again.

Another positive consequence is that there is **no need to create tasks** to fix or improve an issue. As stated before, tasks related to significant issues can be daunting to start. Considering the improvement as part of typical developments, it won't feel like a substantial endeavor, and progress, although small, will always be made towards the end goal.

On the negative side of the consequences, **time management** is one of the main problems. The team might take more time to finish a task since there are more things to consider. It might change the team's velocity, and the product owners and clients might notice a change. Typical tasks that usually had only logic requirements might have some technical specifications to prevent the issues from happening again. But there is always the fact that refactoring code makes future developments a lot easier. Although we might take more time to finish a specific task, future developments will be quicker. But, the team must always discuss if refactoring makes sense at one particular stage. Sometimes it is indeed a priority to abdicate quality for speed.

Another consequence from this solution is the **cluttering of the *DoD***. It might start to be cluttered with small steps to prevent multiple issues that come from the Canvas. Although, in a

way, this is positive for the system quality, for the developer might be hard to consider everything before closing a task or User Story.

An additional consequence is that although this prevents the issue from happening again, the issue persists on older code and needs to be taken care of to eliminate the system's issue. If a team wants to eliminate the process, they have to create tasks to fix the issue on older code incrementally. This still is a problem if the issue is large and complex.

6.4 Example

The start-up where the case study took place was challenged with big technical debt problems since they had a large portion of legacy code. Fixing some of the code quality issues that the teams identified were considerably large endeavors.

During our experience in this start-up, some issues required this type of solution. Since the system had a large quantity of legacy code, the teams had many issues they already know existed but still hadn't formally organized these issues. The Canvas helped them in this aspect, and some of the issues they defined were large and hard to resolve. One of the main issues was related to dead code or code that wasn't being used anymore. With a simple script that analyzed the server logs, they managed to find out that 70% of their core application endpoints were not being called anymore. Removing a large portion of code like this is bound to have a lot of issues, so tackling it in smaller segments was one of the solutions they had, which relates to the SMALL STEPS FOR ANALYTICS pattern referred above. But they also wanted to prevent the issue from growing, so the way they thought to do this was that they should analyze the possibility of generating dead code in new refinements. If this was the case, the portion of code in question should be removed to prevent leaving dead code in the repository.

One more common example is related to code coverage. Some older projects had small code coverage but implementing tests to the sections that didn't have tests yet was very hard since the code was done several years ago, and sometimes it wasn't even done by the same developers working there now. So, since creating more tests was not an option, they created a goal on the canvas that stated that all new code should have 100% coverage. It will naturally increase the code coverage over time since the untested code percentage will reduce with every new code added.

Chapter 7

Conclusions

This last chapter provides a conclusion for this document. First, we provide a quick summary of the results and compare the work analyzed in the State of the Art. Then we provide clear and direct answers to the research questions, and finally, we explore possible work to be done in the future.

7.1 Summary

The conclusions from the state of the art review reveal that software quality models are useful for defining software quality but don't provide an approach for assessing the specified quality attributes in a software project. Software analytics approaches fill this gap by providing a structured approach for operationalizing software quality models. However, they don't explain how software teams should proceed with all this information at hand. Software analytics approaches provide valuable, actionable data. As we described in Section 3.3, continuous improvement approaches provide us exact steps to act with the information acquired by software analytics processes. The Software Analytics Canvas is one of these approaches but doesn't have enough studies around the approach.

This dissertation's main contribution is an industrial case study with two software teams in the context of a start-up company. The start-up teams used the Software Analytics Canvas for ten weeks. Results were analyzed regarding the ease-of-use, usefulness of the approach, awareness of the state of quality of their system, and improvements on the quality of the systems. Two last sessions with the teams took place at the end to understand what changes to the approach should be made. From the results, we were able to understand that the Canvas was a useful approach, easy to understand but harder to use than they had thought. Nevertheless, they confirmed that the approach was valuable to increase their awareness on the state of quality of their system. The approach also helped the teams improve the metrics related to the issues they defined.

After conducting a case study with two teams using the approach we have a better comparison to other techniques investigated in the State of the Art. The Software Analytics Canvas doesn't define what quality is like software quality models do. It's a more flexible approach that focuses on solving the problems that the software teams have regarding code quality. But, although not done

in this case study, they can be used in parallel. Assessment approaches help the teams establish code quality metrics in a structured way but it still follows a models that defines what quality is. The Canvas is a continuous improvement approach, so it focuses on improving the code quality incrementally but doesn't define what quality is. The software teams register the issues that they have regarding to code quality, find measurements and tools that will help them evaluate the state of the issue and then make decisions based on the results obtained.

7.2 Research Answers

In this section, we intend to provide an answer to the three research questions that we defined in Section 4.2.

RQ1: What are the users' perceptions about the usefulness and ease-of-use of the Software Analytics Canvas?

The user's perception was that the Canvas was easier to understand than to use, but the overall sentiment was that the Canvas was useful.

The participants understood the process well and thought it was straightforward. But, even though understanding it was easier than using it, they still felt the Canvas was easy to use.

The usefulness of the approach has two interpretations: the participant perspective and the obtained results. The overall sentiment of the users was that the process was helpful. They said that the approach brought value to them and helped them organize their code quality activities. The managers also thought the approach was relevant for their work, easy to understand, and even though they were less in contact with the Canvas, they said they were more inclined to participate more. The approach also got good results which means that the approach brought value to the teams. We hope that with more time of using the approach the teams will see even more benefits.

RQ2: Are the users more aware of the quality issues of their project, and do they understand the reason behind the metrics being monitored?

Throughout the study the study, the users felt more aware of the quality of the system.

We can also answer this question with the same interpretations as the last question: the participant perspective and the obtained results. As for the participant perspective, although team members said that they had a good enough awareness about the system's quality before the Canvas, it was only intuition. Almost no metrics were being used at the time, and with the new approach, they felt more aware as a new metric would appear, and in the end, they confessed that they were not paying the required attention to code quality and that with the Canvas, their awareness increased.

From the results perspective, some issues came from the investigation of other issues. This means that, with the Canvas, there was the discovery of new issues they weren't aware of before the start of the study. And although they had an idea of what the state of quality was, measuring and documenting these problems helped to be sure of the state of the problem.

RQ3: How does the Software Analytics Canvas affect the software system's quality over time?

Clear progress of the quality of a few issues was observed during the case study.

For the issues chosen, there has been clear progress. The code coverage has been increased on several projects, and dead code is being cleaned up every sprint so that the project becomes less complex and response times of their services APIs are being reduced.

In this experiment, most of the time was dedicated to defining the issues, finding corresponding measurements, and methods and tools to measure the issues. Only in the last sprint of the research has the team started to fix many of the issues as they need to go through thorough planning before being accepted for development. The approach did improve some issues defined, but more research time would have been valuable to give a more precise answer to this question.

7.3 Future Work

There are several aspects that could prove useful to further validate and discuss this work.

- **Conduct this study in other companies** — As we described in Section 5.5, one of the main threats to our results is the fact that the study was only conducted in one company and this company being a startup. Although we got good results, we plan on gathering more evidence of the impact of the approach. Therefore, the study needs to be applied in more start-ups type but also in other types of companies more large like big corporations with even more teams and a more defined structure.
- **Increase the length of the case study** — As we described in this document, we would notice significantly more improvements on the state of quality of the company if the study was conducted for a longer period of time. We did get good improvements in some issues defined in the Canvas but we did not get to see an great impact on the overall quality of the system. This is due to the fact that the teams can't fully concentrate their time to software analytics activities so the progress is slower. There is also some planning activities and formalities slowing the implementation. A longer period of time would be more beneficial to prove the usefulness and the capacity of the approach to improve the system's quality overtime.
- **Conduct the study with other types of teams** — The company where our study took place is a start-up, so there is a small number of engineering teams and they all do everything in terms of technology. Some bigger companies have specialized teams for certain areas like Quality Assurance or DevOps where they would use software analytics more regularly. This would allow them to spend more time with the Canvas and iterate quicker. It would be interesting to see if the approach works for this type of teams as well. Another team that is worth investigating more is the product managers or higher ranks in the companies. It would be interesting to see the impact that it has on these teams and how it affects their decision making.

Appendix A

Surveys

A.1 Kick-off Survey

Software Analytics Survey

Hi!

My name is Duarte Oliveira and I'm studying Software Engineering at FEUP. I'm currently doing my thesis about continuous assessment of code quality through Software Analytics practices in a start-up environment.

Before I present you with my proposal for the next few months, I would like you to answer this questionnaire about the current state of Software Analytics in your team.

***Required**

General

General questions about your role and profession.

1. What's your education level? *

Mark only one oval.

- High School
- Graduate degree
- Master degree
- Doctorate degree

2. What's your role in your current company? *

3. For how many years have you practiced your profession? *

4. For how many years have you been in your current company? *

5. What's your team in your current company? *

6. What's your experience with Agile practices? *

Mark only one oval.

- None
- Novice
- Intermediate
- Expert

Software Analytics

Questions related to Software Analytics and code quality practices.

For the items below with a agree/disagree scale, please state how much you agree with the statement.

7. I am aware of the current state of quality of our software system's code base. *

Mark only one oval.

- 1 2 3 4 5
-
- Strongly Disagree Strongly Agree

8. Can you briefly describe what is Software Analytics to you? *

Four horizontal lines for text input.

Brief Software Analytics definition

From this point forward, we will mention Software analytics as the analytics specific to the domain of software systems. It takes into account source code, static and dynamic characteristics, and although my thesis focuses on code quality, Software Analytics is also related to processes of development and evolution. Software Analytics aims to provide insightful and actionable information in order to improve the efficiency and effectiveness of the software development process.

9. My team has established Software Analytics practices. *

Mark only one oval.

Strongly disagree 1 2 3 4 5 Strongly agree

10. What Software Analytics activities do you practice in your team?

Four horizontal lines for text input.

11. My team is using appropriate Software Analytics tools. *

Mark only one oval.

Strongly disagree 1 2 3 4 5 Strongly agree

12. Which Software Analytics tools is your team using?

Four horizontal lines for text input.

13. My team currently makes use of code quality metrics. *

Mark only one oval.

Strongly disagree 1 2 3 4 5 Strongly agree

14. If you agreed with the previous statement, which ones? How do you use them? (solve problems, as a reference, etc.) Are you aware of the objective you're trying to achieve with these metrics?

Four horizontal lines for text input.

15. The implemented metrics are used in order to solve an existing problem or reach an objective.

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. Did your team establish any quality standards/thresholds for these metrics? For example, in a percentage metric like test code coverage, we could define that the code should have at least 70% code coverage

Mark only one oval.

Yes

No

17. When does your team assess software quality during a sprint? *

Mark only one oval.

Sprint Retrospective

Sprint Review

Sprint Planning

Daily meeting

When implementing a specific feature

Never

Other: _____

18. If you do assess software quality with your team, what topics do you usually discuss with your team?

19. My team frequently includes Software Analytics tasks in the sprint planning. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. What type of tasks do you include?

21. My team dedicates some time of our sprint to refactoring activities. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

22. Are they small refactorings only during your tasks or big refactorings with dedicated tasks?

23. I refactor code during a sprint task in order to deliver better code.

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

24. My team creates refactor tasks in order to improve the code quality incrementally.

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

25. My team assesses the impact of these refactors.

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

This content is neither created nor endorsed by Google.



A.2 Sprint Survey

Survey - Sprint 5

Following up the fifth sprint, I prepared a questionnaire to understand how the Software Analytics Canvas is being perceived. We want to understand the impact the Canvas had on the quality of the system but also, what is the perceived usefulness and ease of use. All the questions refer to the fifth sprint of development.

***Required**

General

General questions about your role and profession.

1. What's your education level? *

Mark only one oval.

- High School
- Graduate degree
- Master degree
- Doctorate degree

2. What's your role in your current company? *

Mark only one oval.

- Team Lead
- Front-end Developer
- Back-end Developer

3. For how many years have you practiced your profession? *

4. For how many years have you been in your current company? *

5. What's your team in your current company? *

Mark only one oval.

- TIMON
- PUMBA

6. What's your experience with Agile practices? *

Mark only one oval.

- None
- Novice
- Intermediate
- Expert

Software Analytics Canvas

Questions about your perception on the Canvas in the previous sprint.

For the items below with a agree/disagree scale, please state how much you agree with the statement.

7. I found it easy to use the Software Analytics Canvas in this sprint. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

8. The Software Analytics Canvas was useful in this sprint. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

9. Following this sprint, I feel that I am more aware of the issues that the company projects have. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

10. The Canvas helped me understand the reasons behind the metrics implemented in this sprint. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

11. The Canvas helps to make sure that the metrics implemented in this sprint are the most relevant to the project. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

12. The Canvas helped improve the code quality of the System in this sprint. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

13. Briefly explain the reason behind your answer to the last question. *

14. The Software Analytics process took a considerable time off this sprint. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

15. The time spent on Software Analytics in this sprint is time well spent and beneficial. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. At this point, I'm confident in my understanding of the concept of Key Issues. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. In this sprint, I found it easy to define Key Issues on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. At this point, I'm confident in my understanding of the concept of Measurements. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. In this sprint, I found it easy to define Measurements on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. At this point, I'm confident in my understanding of the concept of Methods and Tools. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. In this sprint, I found it easy to define Methods and Tools on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

22. At this point, I'm confident in my understanding of the concept of Highlights. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

23. In this sprint, I found it easy to define Highlights on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

24. At this point, I'm confident in my understanding of the concept of Insights. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

25. In this sprint, I found it easy to define Insights on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

26. At this point, I'm confident in my understanding of the concept of Goals. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

27. In this sprint, I found it easy to define Goals on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

28. At this point, I'm confident in my understanding of the concept of Decisions. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

29. In this sprint, I found it easy to define Decisions on the Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

30. What is your opinion on the Canvas? Do you have any comments about the Canvas or the topics of this survey?

This content is neither created nor endorsed by Google.



A.3 Final Assessment Survey - Tech

Final Assessment - Tech

Hi! 🙌

This survey is related to your experience regarding the Software Analytics Canvas in the past few months. We want to compare it to the last methods you were using, but we also want to know about the learning experience, the consequences, and the impact the process had on you and the system.

The following questions are related to the current state compared to the time before using the process.

Please state how much you agree with the statement for the items with a agree/disagree scale.

***Required**

General

General questions about your role and profession.

1. What's your education level? *

Mark only one oval.

- High School
- Graduate degree
- Master degree
- Doctorate degree

2. What's your role in your current company? *

3. For how many years have you practiced your profession? *

4. For how many years have you been in your current company? *

5. What's your team in your current company? *

6. What's your experience with Agile practices? *

Mark only one oval.

- None
- Novice
- Intermediate
- Expert

Before Software Analytics Canvas

7. We had technical debt in our system. *

Mark only one oval.

- 1 2 3 4 5
-
- Strongly disagree Strongly agree
-

8. We were paying the required attention to the system code quality. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

9. By being a start-up, we have pressure on the team to produce value faster and consequently ignore the quality. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

10. What other reasons could have lead to not paying attention to code quality? *

11. The Canvas is an upgrade from my previous software analytics practices. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

Adopting the Software Analytics Canvas

12. I feel that the concept of the Canvas was easy to understand. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

13. I feel that the Canvas was easy to use. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

14. I feel that the Canvas was harder to use than I expected. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

15. I feel that understanding the Canvas was easier than actually using it. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. What made the Canvas easy or hard to understand? *

17. Order the following Canvas lanes from the hardest to the easiest to understand, 1 being the easiest and 7 being the hardest. *

Mark only one oval per row.

	1	2	3	4	5	6	7
Key Issues	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Measurements	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Methods and Tools	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Highlights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Insights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Goals	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Decisions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

18. Order the following Canvas lanes from the less useful to the most useful, 1 being the less useful and 7 being most useful. *

Mark only one oval per row.

	1	2	3	4	5	6	7
Key Issues	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Measurements	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Methods and Tools	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Highlights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Insights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Goals	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Decisions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Consequences

19. We dedicate more time of our sprint to refactoring activities. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. We include more Software Analytics tasks in the sprint planning. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. I feel that I do more refactoring activities during a sprint task to deliver better code. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

22. We create refactor tasks to improve the code quality incrementally. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

23. We assess the impact of these refactors. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

24. How would you describe the impact of the canvas on your refactoring activities? *

25. I am more aware of the current state of quality of our software system's codebase. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

26. How would you describe the impact of the Canvas on your awareness regarding the state of quality of the system? *

27. We are using more appropriate Software Analytics tools. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

28. We are making more use of code quality metrics. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

29. The implemented metrics are more used to solve an existing problem. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

30. How did the Canvas impact the selection process of tools and metrics? *

31. The Software Analytics Canvas process helped the team understand the metrics chosen by first identifying the issues. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

32. What made you better understand the metrics used? *

33. The Canvas helped the team encounter serious issues. *

Mark only one oval.

1 2 3 4 5

Strongly disagree Strongly agree

34. Without the Canvas, some of the issues could have been left unnoticed. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

35. I already knew about the issues defined. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

36. The Canvas helped the team organize technical debt issues in order to take better action on them. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

37. I feel like the Canvas accelerated the process of finding issues and improving the quality. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

38. With this process, we can improve the system quality without harming the value delivered on every sprint. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

39. The Canvas had a positive impact on the company system. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

40. I feel like the Canvas improved the code quality culture of the company. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

41. What do you consider the best consequences of using this process? *

Final Thoughts

42. I want to keep using a Software Analytics process to assess the code quality of the system and improve. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

43. I want that process to be based on the Software Analytics Canvas. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

44. I feel that the process has the potential to help the company improve the overall technical quality of the system. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

45. I think that the Canvas can be improved. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

46. What should the Canvas improve on? Recommend some changes to the process. *

This content is neither created nor endorsed by Google.



A.4 Final Assessment Survey - Managers

Final Assessment - Management

Hi! 🙌

This survey is destined to product managers and other management roles in the company. The objective is to understand your point of view in the usefulness of the approach Software Analytics Canvas, but also if you find the approach easy to understand.

***Required**

General

General questions about your role and profession.

1. What's your education level? *

Mark only one oval.

- High School
- Graduate degree
- Master degree
- Doctorate degree

2. What's your role in your current company? *

3. For how many years have you practiced your profession? *

4. For how many years have you been in your current company? *

5. What's your team in your current company? *

6. What's your experience with Agile practices? *

Mark only one oval.

- None
- Novice
- Intermediate
- Expert

The approach

Software Analytics Canvas is a goal-focused approach to introduce software analytics concepts and practices in an agile development context. The canvas is a hub of information related to software analytics projects where the team can plan their software analytics related activities. It helps teams decide how to address quality issues of the code and architecture and provides transparency over that decision process to all stakeholders. The objective is to ensure that the team members participate in all planning activities, from the definition of the issues to the metrics implemented and results gathered. The team understands the metrics being collected and that these metrics will help address real issues. It's also helpful to avoid monitoring unnecessary measurements since everything is issue-focused.

The Software Analytics Canvas makes use of a board like the ones presented below. HUUB teams have used these for the last four months, and they show a series of issues, the metrics that have been used to generate insights about them, goals established for these metrics, and some decisions made. All the tasks generated from this Canvas were added to Jira.

For the items with a agree/disagree scale, please state how much you agree with the statement.

10. Order the following Canvas lanes from the easiest to the hardest to understand, 1 being the easiest and 7 being the hardest. *

Mark only one oval per row.

	1	2	3	4	5	6	7
Key Issues	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Measurements	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Methods and Tools	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Highlights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Insights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Goals	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Decisions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. Considering your work in particular, order the following Canvas lanes from the less to the most useful, 1 being the less useful and 7 being most useful. *

Mark only one oval per row.

	1	2	3	4	5	6	7
Key Issues	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Measurements	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Methods and Tools	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Highlights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Insights	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Goals	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Decisions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Problems discovered with Software Analytics

12. 70% of the the Core and 90% of the 3PL service endpoints were not being used at all. This is relevant for my work. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

13. Long waiting times can cause a bad experience for the user. Several endpoints take more than 1 second to complete and even some take more than one minute to complete. This is relevant for my work. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

14. Code not covered by tests can be cause of bugs in the system. Some projects had between 50% and 70% of code covered by tests. This is relevant for my work. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

15. These types of findings are important to decision-making when prioritizing tasks for the next sprint. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

Final thoughts

16. During the last 3 months, 15% of the team planned tasks were related to software analytics. I felt that the team performance slowed down. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. I feel like the software analytics process is important and that we should keep using it. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. I feel like dedicating a small percentage of the team tasks to software analytics is beneficial. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. I feel like in my role in the company, understanding the quality of the system is relevant to the decisions I take. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. I feel like participating in the software analytics sessions would be beneficial. *

Mark only one oval.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

This content is neither created nor endorsed by Google.

Google Forms

References

- [1] David E Avison, Francis Lau, Michael D Myers, and Peter Axel Nielsen. Action research. *Communications of the ACM*, 42(1):94–97, 1999.
- [2] Victor R Basili. Software development: A paradigm for the future. In *[1989] Proceedings of the Thirteenth Annual International Computer Software & Applications Conference*, pages 471–485. IEEE, 1989.
- [3] Barry W Boehm, John R Brown, Hans Kaspar, M Lipow, and G McLeod. Merritt.: Characteristics of software quality, 1978.
- [4] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, pages 528–532, 1994.
- [5] Joelma Choma, Eduardo Guerra, Tiago Silva da Silva, Luciana AM Zaina, and Filipe Figueiredo Correia. Towards an artifact to support agile teams in software analytics activities. In *SEKE*, pages 88–122, 2019.
- [6] Joelma Choma, Eduardo Martins Guerra, and Tiago Silva Da Silva. Patterns for implementing software analytics in development teams. In *Proceedings of the 24th Conference on Pattern Languages of Programs*, pages 1–12, 2017.
- [7] Ward Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30, December 1992.
- [8] Yvonne Dittrich, Kari Rönkkö, Jeanette Eriksson, Christina Hansson, and Olle Lindeberg. Cooperative method development. *Empirical Software Engineering*, 13(3):231–260, 2008.
- [9] Farnaz Fotrousi and Samuel A Fricker. Software analytics for planning product evolution. In *International Conference of Software Business*, pages 16–31. Springer, 2016.
- [10] Adrian Furnham. Response bias, social desirability and dissimulation. *Personality and individual differences*, 7(3):385–400, 1986.
- [11] Shrinath Gupta, Himanshu Kumar Singh, Radhika D Venkatasubramanyam, and Umesh Upili. Scqam: a scalable structured code quality assessment method for industrial software. In *Proceedings of the 22nd International Conference on Program Comprehension*, pages 244–252, 2014.
- [12] ISO. Iso. Iec 9126-1: Software engineering-product quality-part 1: Quality model. *Geneva, Switzerland: International Organization for Standardization*, 21, 2001.
- [13] JingFeng Ning, Zhiyu Chen, and Gang Liu. Pdca process application in the continuous improvement of software quality. In *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, volume 1, pages 61–65, 2010.

- [14] Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikładnicki, Nirnaya Tripathi, and Leandro 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)*, pages 75–84. IEEE, 2018.
- [15] Silverio Martínez-Fernández, Andreas Jedlitschka, Liliana Guzmán, and Anna Maria Vollmer. A quality model for actionable analytics in rapid software development. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 370–377. IEEE, 2018.
- [16] Jim A McCall. Factors in software quality. *US Rome Air development center reports*, 1977.
- [17] Reinhold Plösch, Harald Gruber, A Hentschel, Ch Körner, Gustav Pomberger, Stefan Schiffer, Matthias Saft, and S Storck. The emisq method and its tool support-expert-based evaluation of internal software quality. *Innovations in Systems and Software Engineering*, 4(1):3–15, 2008.
- [18] Natalie Robehmed. What is a startup?, Dec 2013.
- [19] Antje Schmitt, Kathrin Rosing, Stephen X Zhang, and Michael Leatherbee. A dynamic model of entrepreneurial uncertainty and business opportunity identification: Exploration as a mediator and entrepreneurial self-efficacy as a moderator. *Entrepreneurship Theory and Practice*, 42(6):835–859, 2018.
- [20] Antje Schmitt, Kathrin Rosing, Stephen X Zhang, and Michael Leatherbee. A dynamic model of entrepreneurial uncertainty and business opportunity identification: Exploration as a mediator and entrepreneurial self-efficacy as a moderator. *Entrepreneurship Theory and Practice*, 42(6):835–859, 2018.
- [21] Japjot Sethi. The differences between entrepreneurs and startup founders, Aug 2014.
- [22] Sandra A Slaughter, Donald E Harter, and Mayuram S Krishnan. Evaluating the cost of software quality. *Communications of the ACM*, 41(8):67–73, 1998.
- [23] Dimitris Stavrinoudis and Michalis Nik Xenos. Comparing internal and external software quality measurements. In *JCKBSE*, pages 115–124, 2008.
- [24] S. Wagner, K. Lochmann, L. Heinemann, M. Kläs, A. Trendowicz, R. Plösch, A. Seidi, A. Goeb, and J. Streit. The quamoco product quality modelling and assessment approach. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1133–1142, 2012.
- [25] Dongmei Zhang, Shi Han, Yingnong Dang, Jian-Guang Lou, Haidong Zhang, and Tao Xie. Software analytics in practice. *IEEE software*, 30(5):30–37, 2013.