sid.inpe.br/mtc-m21d/2021/07.21.11.14-TDI

AN APPROACH TO SUPPORTING AGILE TEAMS IN SOFTWARE ANALYTICS ACTIVITIES

Joelma Choma

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Eduardo Martins Guerra, and Tiago Silva da Silva, approved in June 02, 2021.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/4559H5P>

INPE
São José dos Campos
2021

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21d/2021/07.21.11.14-TDI

AN APPROACH TO SUPPORTING AGILE TEAMS IN SOFTWARE ANALYTICS ACTIVITIES

Joelma Choma

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Eduardo Martins Guerra, and Tiago Silva da Silva, approved in June 02, 2021.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/4559H5P>

INPE
São José dos Campos
2021

Cataloging in Publication Data

Choma, Joelma.

C454a An approach to supporting agile teams in software analytics activities / Joelma Choma. – São José dos Campos : INPE, 2021.

xxii + 178 p. ; (sid.inpe.br/mtc-m21d/2021/07.21.11.14-TDI)

Thesis (Doctorate in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2021.

Guiding : Drs. Eduardo Martins Guerra, and Tiago Silva da Silva.

1. Software analytics. 2. Agile software development. 3. Agile teams. 4. Software quality. 5. Software process improvement. I.Title.

CDU 004.4'413

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DEFESA FINAL DE TESE DE JOELMA CHOMA
BANCA Nº 169/2021

No dia 02 de junho de 2021, as 13h, por teleconferência, o(a) aluno(a) mencionado(a) acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O(A) aluno(a) foi APROVADO(A) pela Banca Examinadora, por unanimidade, em cumprimento ao requisito exigido para obtenção do Título de Doutora em Computação Aplicada. O trabalho precisa da incorporação das correções sugeridas pela Banca Examinadora e revisão final pelo(s) orientador(es).

Título: "An approach to supporting agile teams in software analytics activities "

Eu, Reinaldo Roberto Rosa, como Presidente da Banca Examinadora, assino esta ATA em nome de todos os membros, com o consentimento dos mesmos.

Dr. Reinaldo Roberto Rosa - Presidente - INPE
Dr. Eduardo Martins Guerra - Orientador - INPE
Dr. Tiago Silva da Silva - Orientador - UNIFESP
Dr. Pedro Ribeiro de Andrade - Membro Interno – INPE
Dr. Alan James Peixoto Calheiros - Membro Interno – INPE
Dr. Uirá Kulesza - Membro Externo – UFRN
Dr. Alfredo Goldman vel Lejbman - Membro Externo - USP

sei! assinatura eletrônica

Documento assinado eletronicamente por **Reinaldo Roberto Rosa**, **Pesquisador Titular**, em 10/06/2021, às 16:57 (horário oficial de Brasília), com fundamento no art. 6º do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.mctic.gov.br/verifica.html, informando o código verificador **7517537** e o código CRC **3AC94F78**.

Referência: Processo nº 01340.003623/2021-89 SEI nº 7517537

“I never am really satisfied that I understand anything; because, understand it well as I may, my comprehension can only be an infinitesimal fraction of all I want to understand about the many connections and relations which occur to me, how the matter in question was first thought of or arrived at.”

Ada Lovelace

To my beloved Hélio,
my beautiful Family,
and my dear Friends!

ACKNOWLEDGEMENTS

I thank God for all the blessings in my life, for always guiding and illuminating my way. I also thank my guardian angels for all their protection and inspiration.

To my parents Alexandre and Expedita (in memoriam) for their unconditional and eternal love. To my beloved husband Hélio for all his love, care, and encouragement during my journey, for being at my side at all times. To my sister and my nephews who have always supported me during this endeavor. I also thank my brother and other members of my family for understanding my absence during these years.

I am deeply grateful to Prof. Eduardo Guerra and Prof. Tiago Silva for trusting in my work, for guidance, for stimulus, and for the good mood always. Thanks for the opportunity to work together, it has been a lot of fun.

My eternal gratitude to my dear professor Luciana Zaina, who was directly responsible for encouraging my doctorate and my career as a researcher, having the unforgettable initiative of taking me by the hand to speak with the right people.

To professors and professionals at the National Institute for Space Research (INPE) for their contributions during the conduct of this doctoral. In particular, I would like to thank Prof. Reinaldo Rosa (CAP), Prof. Nandamudi Vijaykumar (CAP), Prof. Alan Calheiros (CAP), Pedro Andrade (CCST), Marcelo BaniK (CEA), Leandro Guarino (CPTEC), and Luiz Eduardo Guarino (CPTEC). Also, to professor Alfredo Goldman at USP for his contributions and suggestions during my doctoral proposal.

To the Applied Computing Laboratory of the INPE, to the User Experience and Usability Laboratory of the Federal University of São Paulo (UNIFESP), and to all the people who participated in the evaluation studies and workshops. My special thanks to Kelson Silva, Vanessa Albuquerque, Vinícius Paschoal, Danilo Alves, Igor Kolesnikov, Allan Silva, Phyllipe Lima, Eduardo Pereira, and Marco Nardes.

To PLoP shepherds and writers workshops participants for all feedback and suggestions to improve my work. Special thanks to Filipe Correia, Rebecca Wirfs-Brock, Joseph Yoder, Lise Hvatum, Ademar Aguiar, Andreas Seitz, and Hernan Astudillo.

To Coordination for the Improvement of Higher Education Personnel (CAPES) for financial support.

ABSTRACT

Software analytics is a data-driven approach to decision making, which allows software practitioners to leverage valuable insights from software data in order to achieve a higher development process productivity and improve several aspects of the software quality. Although widely adopted by large companies, software analytics has not yet reached its full potential for broad adoption. For small teams, such as the ones present on INPE, software analytics is an open question and rarely addressed. In an agile software development context, small teams do not usually use software data to inform their decisions. Often, they make decisions based on feelings and intuitions, which can lead to wasted resources and increase the cost of building and maintaining the software. Unlike most software analytics studies that focus more on analytical tools and techniques, this thesis focuses on how agile teams can add software analytics activities systematically and continuously along with development tasks. As contributions, this research work includes (i) a software analytics pattern language to encourage software practitioners to incorporate data-based approaches to make better decisions in their projects; and (ii) a software analytics canvas modeled from the pattern language as an artifact to support agile teams during the planning and implementation of software analytics activities. An empirical study based on real issues from the EMBRACE project was undertaken to (i) evaluate the proposed canvas under the lens of cognitive activities, using resource model and sequential analysis; and (ii) gather enhancements suggestions using participatory design. From the study results, the canvas was refined and a new version was presented to software practitioners from INPE and other companies.

Keywords: Software Analytics. Agile Software Development. Agile Teams. Software Quality. Software Process Improvement. Software Measurement.

UMA ABORDAGEM DE APOIO ÀS EQUIPES DE DESENVOLVIMENTO ÁGIL EM ATIVIDADES DE ANÁLISE DE SOFTWARE

RESUMO

Software analytics é uma abordagem orientada a dados para a tomada de decisão que permite que os profissionais de software aproveitem *insights* valiosos de dados de software a fim de atingir uma maior produtividade no processo de desenvolvimento e melhorar vários aspectos da qualidade do software. Embora amplamente adotada por grandes empresas, o software analytics ainda não atingiu todo o seu potencial para ampla adoção. Para equipes pequenas, como as presentes no INPE, o software analytics é uma questão aberta e raramente abordada. Em um contexto de desenvolvimento ágil de software, pequenas equipes geralmente não usam dados de software para informar suas decisões. Freqüentemente, eles tomam decisões com base em sentimentos e intuições, o que pode levar ao desperdício de recursos e aumentar o custo de construção e manutenção do software. Ao contrário da maioria dos estudos de software analytics que se concentram mais em ferramentas e técnicas analíticas, esta tese se concentra em como as equipes ágeis podem adicionar atividades de software analytics de forma sistemática e contínua junto com as tarefas de desenvolvimento. Como contribuições, este trabalho de pesquisa inclui (i) uma linguagem de padrão de software analytics para encorajar os profissionais a incorporar abordagens baseadas em dados para tomar melhores decisões em seus projetos; e (ii) o software analytics canvas modelado a partir da linguagem padrão como um artefato para apoiar equipes ágeis durante o planejamento e implementação de atividades de software analytics. Um estudo empírico baseado em questões reais do projeto EMBRACE foi realizado para (i) avaliar o canvas proposto sob a lente de atividades cognitivas, usando modelo de recursos e análise sequencial; e (ii) reunir sugestões de melhorias usando design participativo. A partir dos resultados do estudo, o canvas foi refinado e uma nova versão foi apresentada aos profissionais de software do INPE e de outras empresas.

Palavras-chave: Analytics de Software. Desenvolvimento de Software Ágil. Equipes Ágeis. Qualidade de Software. Melhoria de Processo de Software. Medição de Software.

LIST OF FIGURES

Page

LIST OF TABLES

CONTENTS

Page

1 INTRODUCTION

1.1 Background

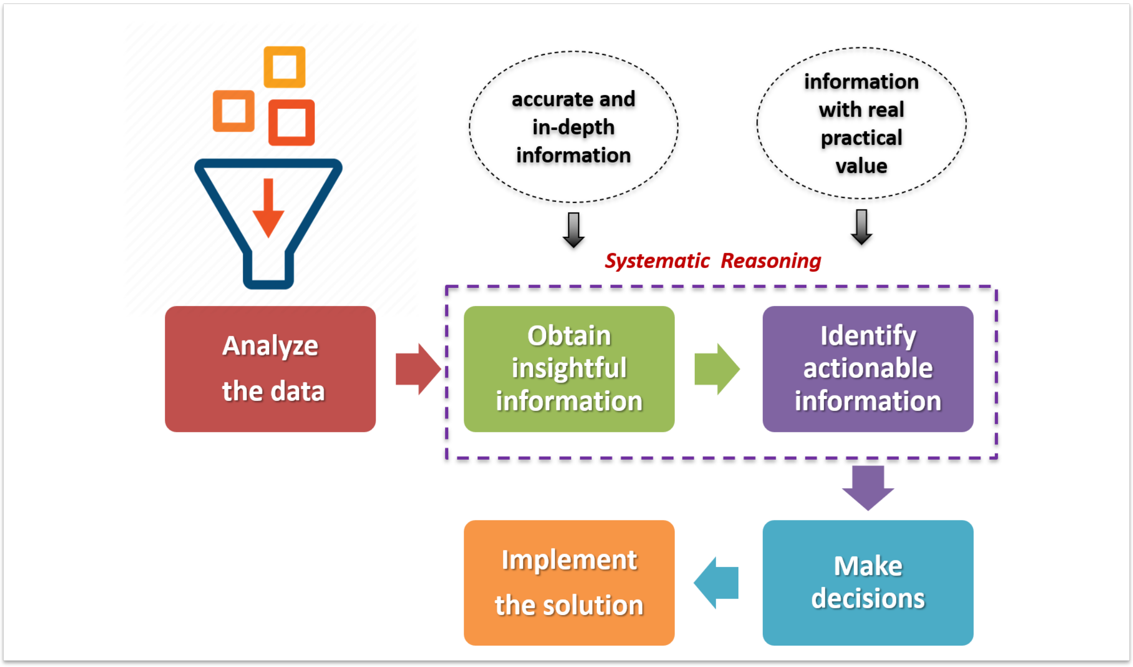
Companies have increasingly invested in *data analytics* solutions to leverage accurate and in-depth information about their business to make better decisions. *Analytics* refers to the extensive use of data, analysis, and systematic reasoning to drive decisions and actions (MAALEJ et al., 2016b). Analytics tools and methods encompass statistical and quantitative analysis, explanatory and predictive models and fact-based management (DAVENPORT; HARRIS, 2007).

Data-driven approaches are adopted mainly as a driver for competitive advantage in several organizational contexts (LISMONT et al., 2017). In the Marketing area, professionals have long used analytics to reach and better understand customers in different consumer markets (DAVENPORT, 2009). Another well-known example is the Web analytics used for site optimization, customer retention, and increase conversion rates. Nowadays, there are various tools available to assist in monitoring Web traffic and analyzing data on user behavior (CHAFFEY; PATRON, 2012).

More recently, *data analytics* have been encouraged as supporting decision making within the context of software development to improve the development processes, as well as the quality of software systems. In 2010, researchers and professionals in the software industry already envisioned the use of *software analytics* bearing in mind the challenges of deploying this approach in such a complex environment – referring both to the complexity of project management and the complexity of the code itself (BUSE; ZIMMERMANN, 2010). In 2011, Zhang et al. (2011) coined the term *"software analytics"* with the purpose to expand the scope of previous work on analytics for software development (BUSE; ZIMMERMANN, 2010) (HULLETT et al., 2011) and on software intelligence (HASSAN; XIE, 2010).

Software analytics (SA) is defined as a data-driven approach that encompasses monitoring, analysis, and understanding of software data to support the decision-making process throughout the different phases of the software lifecycle (ZHANG et al., 2011). The goal of *software analytics* is the use of data analysis and systematic reasoning to leverage insightful information (accurate and in-depth) and actionable (with real practical value) to assist software practitioners (developers, testers, designers, and managers, to name a few) in decision making and completing various tasks concern to systems, users, and development processes. Figure 1.1 summarizes this process.

Figure 1.1 - Information and insights from software analytics.


accurate and in-depth information
information with real practical value
Systematic Reasoning
Analyze the data
Obtain insightful information
Identify actionable information
Make decisions
Implement the solution

SOURCE: Prepared by the author.

For instance, software practitioners could obtain valuable information and insights from data sets aiming at achieving higher development process productivity, improve many aspects of the software quality, or provide a good user experience (MAALEJ et al., 2016a). SA has a quite broad scope of application within the context of software development. Development practices, testing methods, bug treatment, team collaboration, productivity, components reuse, customer requirements, software maintenance, software evolution, quality evaluation, software usage, and services are typical issues that can be addressed with SA (STOREY, 2016). To solve such issues, an SA project can involve analyzing software data retrieved from different sources (e.g. source code, software requirements specifications, bug reports, commit history, test cases, log files, and user feedback), and comes in various forms (both structured and unstructured) (SHULL, 2014).

According to Hassan e Xie (2010), most of these software data are recorded and kept in three categories of software repositories: (i) historical repositories that record information about the evolution and progress of a project (e.g., source control repositories, and bug repositories); (ii) runtime repositories that record log data (e.g., execution traces and user interaction logs); and (iii) code repositories that contain the source code versions of various software systems.

Several studies have been published in *software analytics* area in the last decade. Researchers and practitioners have applied *software analytics* in issues regarding quality of source code involving bug proneness, number of defects, and amount of effort to fix bugs (CERULO et al., 2015) (GUO et al., 2016) (SUREKA et al., 2015); development process including productivity and ROI (RAMARAO et al., 2016) (WANG et al., 2011) (ROBBES et al., 2013) (RAMARAO et al., 2016); business success regarding usage of features, data quality, and user satisfaction (PACHIDI et al., 2014) (GONZÁLEZ-TORRES et al., 2013b); and software properties such as performance, number of transactions, and error log (MUSSON et al., 2013) (AALST, 2015).

For some time now large companies – such as Microsoft, Mozilla, and Google – have been used analytics in software data for better decision making concerning many aspects of software quality and its development processes (BUSE; ZIMMERMANN, 2010). Typically, these companies can afford to keep a dedicated team of researchers, analysts and data scientists with skills to implement *software analytics* projects, working exclusively in the research and development of decision support tools to meet development teams with particular needs, mainly focused on technical indicators at the level of source code, versions, and defects (KIM et al., 2016).

When deploying *software analytics* in their processes, some companies ended up building their own analytics platforms (on-demand) involving different analytics technologies, such as e.g., data mining, machine learning, and data visualization (BAYSAL et al., 2013a). Based on experiences of technology transfer at Microsoft, Zhang et al. (2011) described four steps to implement a *software analytics* project: (i) defining the target-task that will be assisted by SA; (ii) preparation for collection of data to be analyzed; (iii) design and implementation of analytic technologies according to problem formulation, which involves enough information on the system, in-depth knowledge about the data, and algorithms; and (iv) deployment and feedback gathering on analytics implementation involving the domain experts.

1.2 Problem description

Most of the studies in the *software analytics* area have investigated and proposed methods, techniques and tools for assisting those who make critical decisions in software projects (ABDELLATIF et al., 2015) (ANWAR; PFAHL, 2017). However, to the best of our knowledge, there is no consolidated approach to supporting software practitioners in planning and implementing *software analytics* activities in their projects. Despite widely adopted by large companies, *software analytics* has not yet reached its full potential for broad industrial adoption.

For small software companies, for example, *software analytics* is an open question and rarely addressed. Small companies often have fewer resources to include data measurement programs in their projects. In certain cases, the effort to implement *software analytics* is taken into account whether is worthwhile (ROBBES et al., 2013). Traditionally, small teams do not have professionals with skills to plan and develop *software analytics* activities. When configuring measurement tools, software practitioners often collect data beyond what they really need and, sometimes are not able to take advantage of advanced analytical applications, which include trend analysis, classification algorithms, predictive modeling, statistical modeling, optimization and simulation, and data mining (SHULL, 2014) (BUSE; ZIMMERMANN, 2010). That issue is particularly relevant to teams that use agile software development (BECK et al., 2001) and that rarely succeed in establishing measurement programs due to the urgency of the product delivery, lack of time or other constraints (LIECHTI et al., 2017b). Overall, assessment and measurement modes require to be compatible with Agile values and principles that emphasize "working software is the primary measure of progress", over measuring intermediate work products (BECK et al., 2001).

In Agile contexts, the measurement tends to be immediate and straightforward. On the other hand, another Agile principle claims that "continuous attention to technical excellence and good design enhances agility". Technical excellence is about uncovering better ways of developing software, and process improvement is one of the main reasons for measurement in Agile software development (HARTMANN; DYMOND, 2006). In this sense, *software analytics* can support agile teams to make more appropriate changes based on actual data, rather than only on their personal experiences or intuitions. Additionally, the adoption of data-driven continuous improvement can help agile teams to save resources and decrease the cost of building and maintaining the software (HASSAN; XIE, 2010).

In the context of the National Institute for Space Research (INPE), *software analytics* is aligned with the quality management and productivity in software development in the space research areas, such as meteorology, remote sensing, space weather, image processing, astrophysics, and aerospace engineering. In general, the software development teams at these research centers are small and most of them typically adopt agile practices. Talking to some of these teams, we found that *software analytics* could be used to address different issues within their development processes. At INPE, many software projects that support research or research-related projects often suffer unexpected demands since such projects are guided by results obtained.

During our research project, we held meetings with some development teams within INPE and identified some issues regarding *software analytics*. We raised these issues with members from the EMBRACE team at the Space Weather Center, the TerraME team at the Earth System Science Center, and the CPTEC team at the Weather Forecast and Climate Studies Center in Cachoeira Paulista.

For the EMBRACE team, *software analytics* could help them maintain and evolve legacy software involving web applications and sensor data. Some questions raised by them were: How much invalid data do we have in our database? Which products are most accessed or used from our portal? What are the major maintenance troubles? For the TerraME team, *software analytics* could be applied to explore the usage data of API developed for dynamical modeling and geo-spatial data visualization. In geo-spatial research, Application Programming Interfaces (APIs) are very useful in the data visualization process. But, these applications are not always easy to use (PICCIONI et al., 2013), mainly by scientist users who are not experts in software engineering. A key issue addressed to *software analytics* could be: What are the factors that may be influencing the API's usability? For the CPTEC team, *software analytics* could also be used to support them in making decisions about maintaining and improving their products. Some of the questions initially raised by the team were: What is the profile of our users? What are the bugs existing in the applications' features? What content is most accessed by users? Which applications are accessed and used most often? How much are our applications consuming computational resources?

1.3 Research questions and objectives

The main objective of this thesis is *an approach to supporting agile teams in software analytics activities*. To achieve this goal, we have outlined three research questions:

RQ1) What is the state of the art in the area of *software analytics* and the research gaps? What kind of issues are commonly addressed by researchers and software professionals in this area?

RQ2) What good practices could assist software professionals in conducting *software analytics* projects?

RQ3) How could the planning and management of *software analytics* activities be supported in practice?

This thesis studied how to support small and medium agile teams during the planning and implementation of *software analytics* activities. The steps of this study are 1) know how the research in the area of *software analytics* is doing and what types of problems have been commonly addressed in this area; 2) identity which good practices can be adopted by professionals who wish to conduct *software analytics* projects to reduce the cost of uncertainty and change; and finally 3) to propose and evaluate an approach to support software practitioners from planning and implementing *software analytics* activities to informed decision making by insightful and actionable insights. To answer the research questions, the following research objectives are formulated:

- To investigate studies about *software analytics* including identifying research gaps and the common issues addressed by researchers and software practitioners in this area.
- To identify from the existing literature a set of good practices to assist software practitioners in *software analytics* projects.
- To design an artifact to support software practitioners throughout the *software analytics* process.
- To evaluate the use of the artifact from the perspective of software practitioners.
- To refine the proposed artifact from evaluation findings.

1.4 Theme relevance and contributions

Research in the *software analytics* area is highly relevant both for academia and industry. At the intersection between academia and practice, *software analytics* is fundamentally an empirical approach that addresses real problems from the software industry – e.g., assessing architecture and software design, allocating resources and estimating effort effectively, comparing technologies, evaluating user experience. Thus, the potential issues identified in an industrial context involve software data analysis from actual projects in order to produce insightful results for practice. For academia, this is an excellent opportunity to employ techniques and methods from a grounded knowledge base – e.g., data science, data mining, descriptive and statistical analysis, predictive and prescriptive algorithms.

In this context, our research project aims to help and encourage software practitioners to make better data-driven decisions by an approach that leads them to an informed decision-making process. The main contributions of this proposal include:

- a literature review to identify research gaps in the *software analytics* area.
- a survey of the state of the practice concerning how the metrics are used by software practitioners for decision making in agile development contexts.
- a set of patterns to encourage agile practitioners to implement a *software analytics* approach in their software projects.
- artifacts (techniques and methods) to support researchers and practitioners to introduce *software analytics* in their development process.

Unlike the majority of *software analytics* studies that focus more on analytical techniques and tools, our research is concerned to investigate how agile teams could add *software analytics* activities along with development tasks systematically and continuously. This research has practical applicability and is closely linked to process improvement and software quality, focusing on software practitioners working with agile practices. We recognize that large software companies could offer fewer barriers to the adoption of *software analytics* as part of their processes. Also, most of them have better condition to hire a specialist team of professionals like researchers, analysts, and data scientists to conduct *software analytics* projects and develop their analytical tools. However, the foremost challenge is the *software analytics* culture adoption for small and medium-sized teams with few resources to invest in new practices. Given this challenge, our approach has been designed to meet the needs of small and medium-sized teams by requiring minimal effort to fit *software analytics* activities in the teams' daily work. In this sense, the results of this research are also relevant for INPE that generally have small teams adopting agile methods. As raised with some INPE teams, several important issues can be resolved and dealt by using the developed approach.

1.5 Chapters overview

The chapters of this thesis proposal are arranged based on the research design. The remainder of this document is organized into the following chapters:

Chapter 2 contains the systematic literature review on the state of art of *software analytics* including a mapping of research gaps and issues commonly addressed.

Chapter 3 outlines the research methodology, including the research design and the activities of the study. The research design describes the study's characteristics in terms of purpose, methods of collecting, analysis, and the flow of research activities used to achieve the research objectives.

Chapter 4 presents a set of patterns for lightweight and interactive *software analytics* projects, identified from experiences reports selected from the previous literature review study.

Chapter 5 describes the approach proposed in the canvas format to support agile teams during planning and implementing *software analytics* activities.

Chapter 6 covers the first round of evaluation of *software analytics* canvas which involves an observational study undertaken in a controlled setting.

Chapter 7 introduces the new version of *software analytics* canvas and second round of evaluation from a case study and workshops with software practitioners.

Chapter 8 presents the conclusions of this work and some consideration on the research questions, highlights the contributions and directions for future works stemming from this study.

2 LITERATURE REVIEW ON SOFTWARE ANALYTICS

This chapter presents the literature review on the state of the art in the area of *software analytics* carried out to discover the existing research gaps and identify the kind of issues that are commonly addressed by researchers and software professionals in this area (RQ1). The results of the systematic mapping conducted on the literature provided us an overview of the area of *software analytics* including experience reports that will serve as a basis to delineate and propose the language pattern presented in Chapter 4.

Section 2.1 outlines the review method adopted. Sections 2.2 presents the results of the systematic mapping driven to answer the predefined research questions about the domain under investigation. Section 2.3 highlights the findings and research gaps based on the systematic mapping results. And, Section 2.4 discusses the study limitations and threats to validity.

2.1 Research method

To systematically identifying and aggregating the evidence about the *software analytics* area, the research method consisted of a systematic mapping. Mapping study (also referred to as a scoping study) is usually employed in early research stages to provide an overview of the types of research and results available within certain interest area (KITCHENHAM; CHARTERS, 2007). The focus of mapping studies is on classification and thematic analysis since the related questions are concerned about how the researched area is structured (PETERSEN et al., 2015).

To achieve rigor in our mapping study, we considered the guidelines proposed by Petersen et al. (2015). Based on their guidelines, we performed the following steps: (i) definition of research questions; (ii) definition of a search strategy and data sources; (iii) definition inclusion and exclusion criteria; (iv) definition of quality assessment criteria; (v) classification scheme and data extraction; and (vi) data analysis and synthesis of evidence.

2.1.1 Research question

The goal of this literature review is to identify studies about the use of *software analytics* for decision making in the software development context. After the selection of the studies, we first map them into the classification scheme to identify research gaps and current trends in the *software analytics* field.

After that, we analyzed in detail and synthesized the results to identify the main issues commonly addressed in *software analytics* projects, including methods, metrics, measurements, and tools commonly used. Table 2.1 presents the questions addressed to the mapping study (MQs) and the rationals for every question.

Table 2.1 - Questions addressed to mapping study.

Mapping questions	Rational
MQ1: How is the frequency of published studies over the years? Which are the primary publication venues?	This analysis indicates how *software analytics* research is evolving over the years in terms of the number of papers published and the leading forums for publication of future studies.
MQ2: Which are the research types of primary studies and the most frequently applied research methods?	The answer to this question allows the traceability of studies, shows research trends over time, and can point out the lack of specific contribution types.
MQ3: What has been the research domain over the years and types of contributions for *software analytics*?	The answer to this question shows research focus on a specific area over the years and allows the identification of research gaps within the *software analytics* area.
MQ4: Which are the chain of co-authorship within *software analytics* research?	This analysis allows the identification of the co-authorship networks and future opportunities for partnerships with researchers in the *software analytics* area.
MQ5: What the main issues addressed in *software analytics* research?	By categorizing the issues addressed in the area of *software analytics*, the research gaps can also be identified.

SOURCE: Prepared by the author.

2.1.2 Search strategy and data sources

The search string was defined with keywords extracted from five relevant and well-know articles related to *software analytics* (MENZIES; ZIMMERMANN, 2013; HASSAN; XIE, 2010; BUSE; ZIMMERMANN, 2012; ZHANG et al., 2011; BUSE; ZIMMERMANN, 2010). Before defining our search string, some attempts have been made by combining keywords in different ways. However, some options were discarded because they returned a very limited number of articles. In the final test, in addition to returning a balanced amount of articles, we verified that the defined string included in the search result the set of studies chosen as the key articles related to the area of interest, which ensured the quality of the search.

The general search query was formulated as showed in Table 2.2.

Table 2.2 - Search string.

("software analytics" OR "software intelligence") AND ("software measurement" OR "data-driven decision" OR "decision making" OR "software metric" OR "mining software repositories" OR "software process" OR "software project" OR "software development" OR "user experience")

SOURCE: Prepared by the author.

As for strategy, the search for primary studies was conducted in two rounds. In the first round, we performed an automated search on seven databases considered relevant in the area of software engineering and computer science (KITCHENHAM; CHARTERS, 2007): IEEE Xplore, ACM Digital Library, Scopus, SpringerLink, Web of Science, Engineering Village, and Science Direct (Elsevier). We used the defined search string in every one of those databases for searching in titles, abstracts, and keywords. Additionally, we set the publication date parameter to retrieve papers from 2009. The automated search was performed in the period between February 14-18, 2019.

In the second round, we performed a search in Google Scholar using a forward snowballing approach (WOHLIN, 2014) by searching in the reference list of the five relevant papers (MENZIES; ZIMMERMANN, 2013; HASSAN; XIE, 2010; BUSE; ZIMMERMANN, 2012; ZHANG et al., 2011; BUSE; ZIMMERMANN, 2010). The search string was also used at this stage to restrict the number of articles and improve retrieval of the most relevant studies. This search round was performed on February 18, 2019.

2.1.3 Selection criteria and screening

After the removal of the duplicate studies, the studies identified during automated search and snowballing rounds were evaluated based on the inclusion and exclusion criteria as showed in Table 2.3.

2.1.4 Quality assessment

To evaluate the quality of the selected studies, we made use of rigor and relevance scores suggested by Ivarsson e Gorschek (2011).

Table 2.3 - Inclusion and exclusion criteria.

Inclusion criteria	Exclusion criteria
I1: Studies related to the use of *software analytics* in decision making in the area of SE, including practices, methods, or tools.	E1: Technical report, masters, and Ph.D. thesis, and book chapters.
I2: Peer-reviewed research studies published in journals and conference proceedings.	E2: Expanded summaries or summarized keynotes.
I3: Experience report involving issues addressed to *software analytics*.	E3: Tutorial and panels presented in conferences
I4: The full-text of the paper is available.	E4: Studies not presented in English.

SOURCE: Prepared by the author.

Rigor consists of assessing how rigorous was the way the study was presented, while relevance consists in assessing the potential research impact for industry. Rigor is decomposed into three aspects: description of context, study design, and validity threats. These aspects are scored with three score levels: weak (0), medium (0.5), and strong (1). The scoring of each rigor aspect is described in Table 2.4. Relevance is decomposed into four aspects: subjects, context type, scale, and research method. The scoring discerns aspects that contribute to relevance (1) from aspects that do not contribute to relevance (0). Table 2.5 present the description of each relevance aspect.

Table 2.4 - Rigor aspects.

Item	Rigor Aspect	Score	Description
QA1	Context Description	1	The context is described in detail allowing its understanding and comparison.
		0.5	The study context is briefly mentioned.
		0	No description of the context is presented.
QA2	Study Design	1	The study design is described in detail.
		0.5	The study design is briefly described.
		0	No description of the study design is presented.
QA3	Threats to Validity	1	The threats to validity is discussed in detail.
		0.5	The validity of the study is mentioned but not described in detail.
		0	Any threats to validity is discussed.

SOURCE: Prepared by the author.

Table 2.5 - Relevance aspects.

Item	Relevance Aspect	Score	Description
QA4	Subjects	1	Industry professional
		0	Students, researchers, subject no mentioned
QA5	Context Type	1	Industrial setting
		0	Laboratory, context no mentioned
QA6	Scale	1	Realistic size: industrial scale
		0	Down-scaled industrial or toy example
QA7	Research Method	1	Action research, lessons learned, case study, field study, interview, survey
		0	Conceptual or mathematical analysis, laboratory experiments, other, N/A.

SOURCE: Prepared by the author.

2.1.5 Data extraction and classification scheme

The selected studies were managed using Mendeley [1] – a reference management tool – to support the data extraction process. A spreadsheet was used for data collection. Table 2.6 present the form with the data items extracted after full reading the articles. This form items was previously elaborated based on research questions. In our classification scheme, the research method classification is based on the work of Runeson et al. (2012). To classify the type of research, we have used existing classifications suggested by Wieringa et al. (2006) and Petersen et al. (2015).

Below, we provide a short description of the six types of research considered in this study:

- Evaluation: the study refers to implementation in practice including an evaluation of the techniques, methods or tools related with SA. Examples of evaluation studies include action research, ethnography surveys, case studies, field studies, and a controlled experiment with practitioners.
- Solution Proposal: a novel technique to SA (or significant improvement of an existing technique) is proposed, and its relevance is demonstrated by an application in practice, or at least a proof-of-concept may be presented. However, we also include in this category the solutions proposals in an initial phase of research which have not yet been evaluated or validated.

[1] Source: https://www.mendeley.com

- Validation: the study mentions a proposed solution in the SA area, and presents an evaluation of this approach by research methods, such as by experiments, simulation, or prototyping.

- Experience Report: it refers to an author's personal experience. The experience reports explain how techniques, methods, and/or tools of SA have been applied in practice, and also presents a list of lessons learned.

- Opinion: refers to the author's opinion about issues regarding SA field; regardless of the research methodologies or related work.

- Philosophical: these studies outline a new way of looking at existing things, by an idealization of a new conceptual framework in the field of SA.

Table 2.6 - Data extraction form.

#	Data item	Data Description	MQ #
1	ID	Paper ID	Overview
2	Database	Data source from which the study was selected	Overview
3	Title	Title of the paper	
4	Year	The year of publication	Overview
5	Keywords	List of keywords	Overview
6	Summary	The summary of study including aim, method, results, and conclusions	Overview
7	Venue	The name of the venue where the study was published	Overview
8	Publication type	Journal, conferences, workshops	MQ1
9	Research Type	Evaluation, validation, solution proposal, personal opinion, philosophical, experience report	MQ2
10	Research Method	Single case study, multiple case study, experiment, survey, simulation, proof-of-concept, action research, etc.	MQ2
11	Research Domain	Research domain taking into account the list of keywords and the venue where the study was published	MQ3
12	Authors	List of all the authors of the paper	MQ4
13	Contribution Type	data analytics (i.e., techniques, methods and tools); data mining (i.e., method and tools); data collection (i.e., method and tools); prediction models; visual *software analytics*; *software analytics* platform; *software analytics* monitoring (i.e., measure, metrics and indicators); *software analytics* issues and concepts; *software analytics* projects implementation; advice, implication and/or recommendations; and lessons learned.	MQ3
15	SA Issue Type	effort prediction, fault prediction, bug treatment process, teamwork and collaboration, productivity, program comprehension, etc.	MQ5

SOURCE: Prepared by the author.

The research domain was classified base on the list of keywords and the venue where the study was published, while types of contribution refer to the type of intervention being studied involving processes, methods, models, or tools (PETERSEN et al., 2008).

2.1.6 Data analysis

The data analysis was conducted by using descriptive statistics, content analysis, and narrative synthesis for explaining in details and interpreting the findings coming from the content analysis. The content analysis consists of categorizing data and analyzes frequencies of themes within categories (PETERSEN, 2011).

2.2 Mapping results

2.2.1 Primary studies collection

Figure 2.1 presents the results from searching, selection, and screening of the primary studies. The searching string used on seven databases retrieved 382 records, while the snowballing searching retrieved 217 records. After removing duplicates studies, 168 articles were selected from databases and 57 articles from the snowballing process. Afterward, 84 articles were excluded based on the inclusion and exclusion criteria during the screening step. In total, 141 articles remained for review. The list of the selected studies is presented in Appendix A.

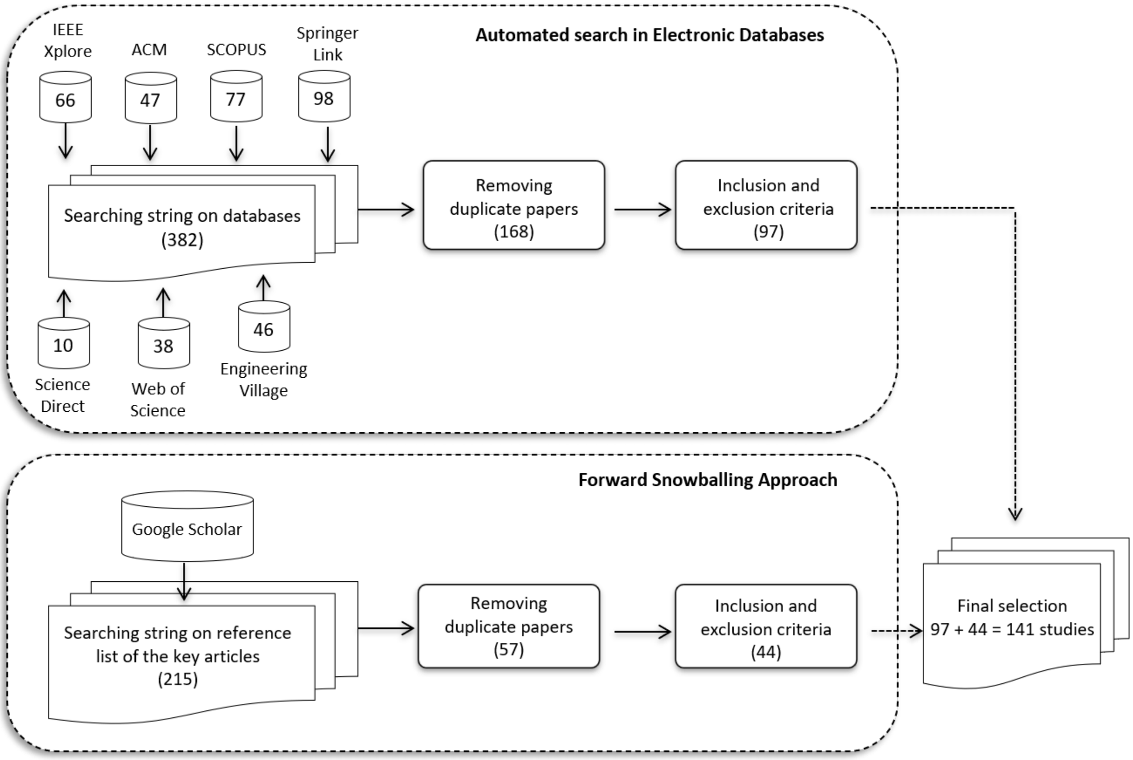
Table 2.7 shows the number of primary studies considered in the search, selection, and screening process per database. Table 2.8 shows the results of forward snowballing from five key papers related to the SA area. As aforementioned, the same search string was applied to select the most relevant studies, which were referencing one or more key papers. During this step, many studies were not selected because they had already been previously selected through searching in the digital libraries.

Table 2.7 - Number of primary studies per database.

Database	Initial Set	Removed	Selected
IEEE Xplore	66	34	32
ACM Digital Library	47	25	22
Scopus	77	64	13
SpringerLink	98	80	18
Web of Science	38	32	6
Engineering Village	46	43	3
Science Direct	10	7	3

SOURCE: Prepared by the author.

Figure 2.1 - Results of study search, selection, and screening process for primary studies.


Automated search in Electronic Databases
IEEE Xplore
66
ACM
47
SCOPUS
77
Springer Link
98
Searching string on databases (382)
10
Science Direct
38
Web of Science
46
Engineering Village
Removing duplicate papers (168)
Inclusion and exclusion criteria (97)
Forward Snowballing Approach
Google Scholar
Searching string on reference list of the key articles (215)
Removing duplicate papers (57)
Inclusion and exclusion criteria (44)
Final selection 97 + 44 = 141 studies

SOURCE: Prepared by the author.

Table 2.8 - Results of forward snowballing performed in key studies.

Paper#	Year	Cited by	Search String	Selected	Reference
P1	2010	79	44	13	Hassan e Xie (2010)
P2	2010	40	27	5	Buse e Zimmermann (2010)
P3	2011	49	29	4	Zhang et al. (2011)
P4	2012	157	52	10	Buse e Zimmermann (2012)
P5	2013	96	63	12	Menzies e Zimmermann (2013)

SOURCE: Prepared by the author.

2.2.2 Quality evaluation of the primary studies

To evaluate rigor and relevance aspects of the selected studies, we used the quality assessment criteria presented in Section 2.1.4 (Tables 2.4 and 2.5). On the whole, we evaluated the quality of 102 papers (72.3%) matching to case study, experiment, lessons learned, action research, and surveys.

The quality of the other studies was not assessed because they referred to research agendas, personal opinion, philosophical articles and demos. It is noteworthy that although they were not evaluated in this step, they were still remained in our review.

Table 2.9 presents quality evaluation concerning rigor aspects. For each research method, we averaged the scores of each rigor aspect (see Table 2.4) and the average of the final score, including all aspects. Overall, the studies had a low rating in the aspect refer to the discussion of threats to validity (QA3), because in many of them this aspect had not been described in detail, or it was not mentioned. Lessons learned presented the lowest rigor score, while surveys, experiments, action research, and case studies attended the rigor aspects at a higher level.

Table 2.9 - Quality evaluation of the rigor aspects.

Research Method	N	QA1	QA2	QA3	Score
Case Study	54	0.82	0.68	0.53	2.03
Experiment	26	0.96	0.92	0.75	2.63
Lessons Learned	16	0.78	0.28	0.06	1.13
Action Research	3	1.00	1.00	0.33	2.33
Survey	3	1.00	1.00	0.83	2.83

SOURCE: Prepared by the author.

Table 2.10 presents quality evaluation concerning aspects of industrial relevance. For each research method, we also averaged the scores of each aspect (see Table 2.5) and the average of the final score, including all aspects. Action research and surveys obtained a maximum score concerning relevance, while the other three methods representing 68% of the studies obtained a score above 3 points.

Table 2.10 - Quality evaluation of the relevance aspects.

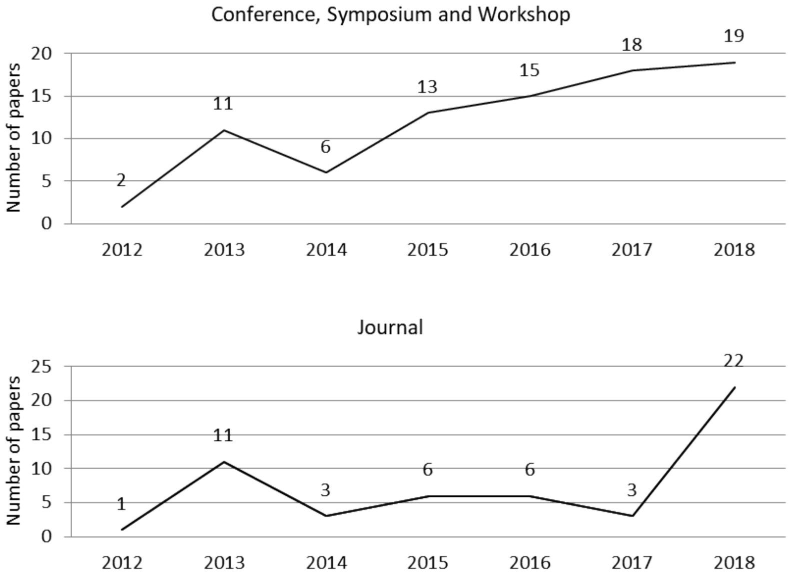
Research Method	N	QA4	QA5	QA6	QA7	Score
Case Study	54	0.70	0.76	0.98	1.00	3.44
Experiment	26	0.62	0.62	1.00	0.81	3.05
Lessons Learned	16	0.75	0.88	0.94	0.94	3.51
Action Research	3	1.00	1.00	1.00	1.00	4.00
Survey	3	1.00	1.00	1.00	1.00	4.00

SOURCE: Prepared by the author.

2.2.3 Frequency and publication venues (MQ1)

Figure 2.2 shows the number of primary studies published from 2012 to 2018. Note that the graphs do not include the studies published in 2019, once only two months (January-February of 2019) were covered in the search period. The number of studies related to *software analytics* published in conferences, symposium, and workshop has increased year by year in the last four years. The number of studies published in journals was smaller than in the other venues until 2017. Nevertheless, the number of publications in journals had a peak in the last year, exceeding the other venues.

Figure 2.2 - Year-wise distribution of studies.



SOURCE: Prepared by the author.

Table 2.11 shows the distribution of the primary studies by publication venue, where we list only the names of venues with more than two publications. From the selected studies, 40.4% are journals, while international conferences, symposiums or workshops, account for 59.6%. The main forum for publications was the journal IEEE Software. The second one was the journal Empirical Software Engineering. Thirdly, the International Workshop on Software Analytics is the leading conference that has been taking place annually since 2015 (first edition).

Table 2.11 - Publication venues.

Journal	N	%
IEEE Software	18	12.8
Empirical Software Engineering	13	9.2
IEEE Transactions on Software Engineering	5	3.5
Science of Computer Programming	4	2.8
Software Quality Journal	3	2.1
International Requirements Engineering	2	1.4
Others	12	8.5
Total	57	40.4
Conference. Symposium and Workshop	N	%
Workshop on Software Analytics	11	7.8
International Conference on Software Engineering	9	6.4
Conference on Automated SE	5	3.5
Joint Meeting on Foundations of SE	5	3.5
Symposium on Foundations of SE	3	2.1
Conference on SE & Knowledge Engineering	2	1.4
Conference on SE Companion	2	1.4
Symposium on Empirical SE & Measurement	2	1.4
Others	43	30.5
Total	84	59.6

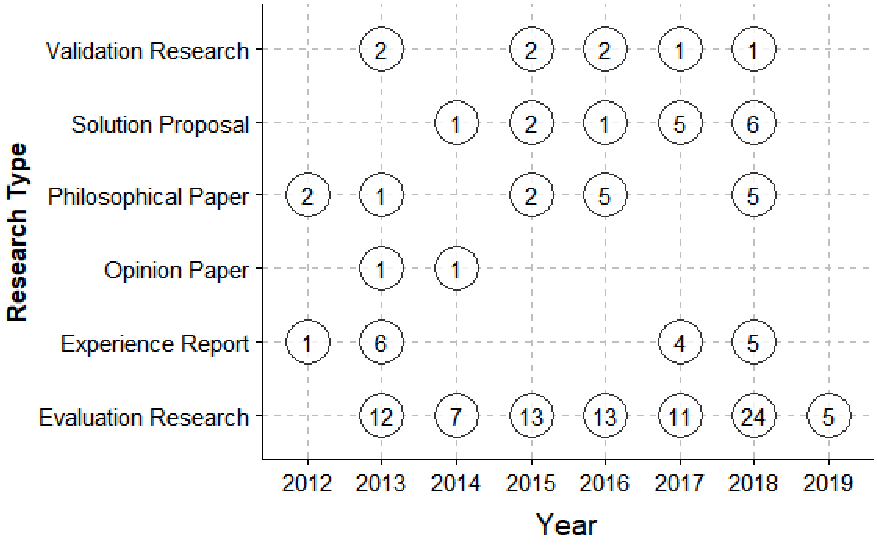
SOURCE: Prepared by the author.

2.2.4 Research type (MQ2)

Figure 2.3 shows the mapping of the primaries studies according to the research type over the years. Most of the studies (85 papers, 60.2%) were categorized as Evaluation Research, representing on average, 12 papers per year. Experience reports and solution proposal represent about 22% of the papers, while opinion and philosophical paper represent only 12% of the total.

Table 2.12 presents the distribution of the studies' underlying research method. Case studies are used by a large part of the studies (54 papers, 38.2%). As shown in the table, 29 studies were reported as Single-Case Study, whilst 25 studies were reported as Multi-Case study. Of the 26 experiments reported, only one study involved students. Lessons learned were reported from 16 experience reports. Few studies conducted action research and surveys (4%), whereas the demos category including studies using simulation, prototyping, proof-of-concept, or use cases represented 9% of studies. For research agendas, personal opinion, and philosophical papers, we did not consider any research method.

Figure 2.3 - Year-wise distribution of studies and research type.



SOURCE: Prepared by the author.

2.2.5 Research focus over time (MQ3)

Figure 2.4 shows the bubble chart with the mapping the primary studies by type of contribution and research domain. The bubbles indicate the numbers of papers in each category over time. Both the domain category and contribution category emerged from the analysis of keywords together with title, abstract, and publication forums. In some cases, reading other parts of the article were also necessary to define such categories.

Software analytics was the research domain with the highest frequency of studies (72 - 51%). The other studies had a main focus on other research domains: *software prediction* (19 - 13.4%), *software maintenance and evolution* (14 - 9.9%), *mining software repositories* (12 - 8.5%), *requirements engineering* (8 - 5.6%), *software quality* (7 - 4.9%), *software testing* (4 - 2.8%), and *data-driven software development* (5 - 3.5%). Although some of the studies do not focus directly on the area of *software analytics*, we have identified some contribution involving our area of interest. We identified and mapped the studies into eleven types of contribution. The top three types of contributions with the highest frequency were *data analysis including techniques, methods, and tools* (32 - 22.6%), *predictive modelling* (29 - 20.5%), and *data mining including methods and tools* (26 - 18.4%).

Table 2.12 - Studies per types of research approach.

Research Method	N	Papers
Multi Case Study	25	S5 - S6 - S14 - S20 - S23 - S33 - S36 - S40 - S41 - S47 - S51 - S54 - S55 - S63 - S69 - S79 - S80 - S93 - S100 - S102 - S109 - S110 - S111 - S131
Single Case Study	29	S11 - S12 - S15 - S22 - S24 - S26 - S34 - S43 - S44 - S45 - S46 - S49 - S58 - S70 - S72 - S73 - S81 - S83 - S89 - S90 - S91 - S103 - S107 - S108 - S114 - S121 - S129 - S136 - S138
Experiment	26	S7 - S28 - S29 - S32 - S35 - S42 - S57 - S66 - S67 - S76 - S78 - S85 - S88 - S96 - S98 - S104 - S106 - S115 - S122 - S123 - S127 - S128 - S137 - S139 - S140 - S141
Lessons Learned	16	S3 - S8 - S10 - S16 - S18 - S19 - S21 - S77 - S82 - S84 - S95 - S101 - S105 - S113 - S118 - S135
Action Research	3	S25 - S56 - S117
Survey	3	S27 - S48 - S116
Demos	13	S4 - S13 - S37 - S38 - S39 - S53 - S62 - S68 - S94 - S97 - S99 - S119 - S126
No Method	26	S1 - S2 - S9 - S17 - S30 - S31 - S50 - S52 - S59 - S60 - S61 - S64 - S71 - S74 - S75 - S86 - S87 - S92 - S112 - S120 - S124 - S125 - S130 - S132 - S133 - S134

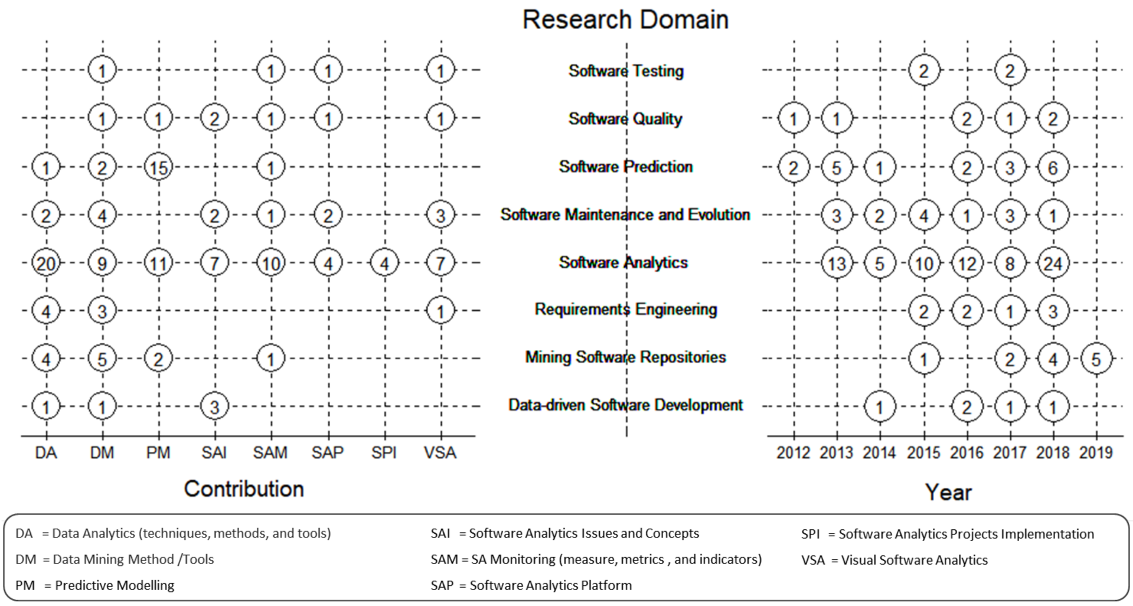
SOURCE: Prepared by the author.

A fewer number of contributions were identified on proposals of some studies on *monitoring in software analytics* encompassing measures, metrics, and indicators (15 - 10.6 %), studies discussing *issues and concepts related to software analytics* (14 - 9,9%), *visual software analytics* approaches (13 - 9.2%), and *software analytics platform* (8 - 5.6 %). Finally, we identified only four studies on the *implementation of software analytics projects.*

2.2.6 Co-authorship network (MQ4)

Figure 2.5 shows the co-authorship graph, which includes 350 nodes representing unique authors and 692 edges representing the relationships among them. We used the Gephi graphing tool (BASTIAN et al., 2009) to create this graph, which is also available at https://bit.ly/2PjBJYy. By analyzing the number of connected components in the graph, we identified 65 sets of authors that are mutually connected through a network of co-authorship. There are 8 major sub-graphs, including more than 10 authors nodes.

Figure 2.4 - Mapping of primary studies by type of contribution and research domain over time.



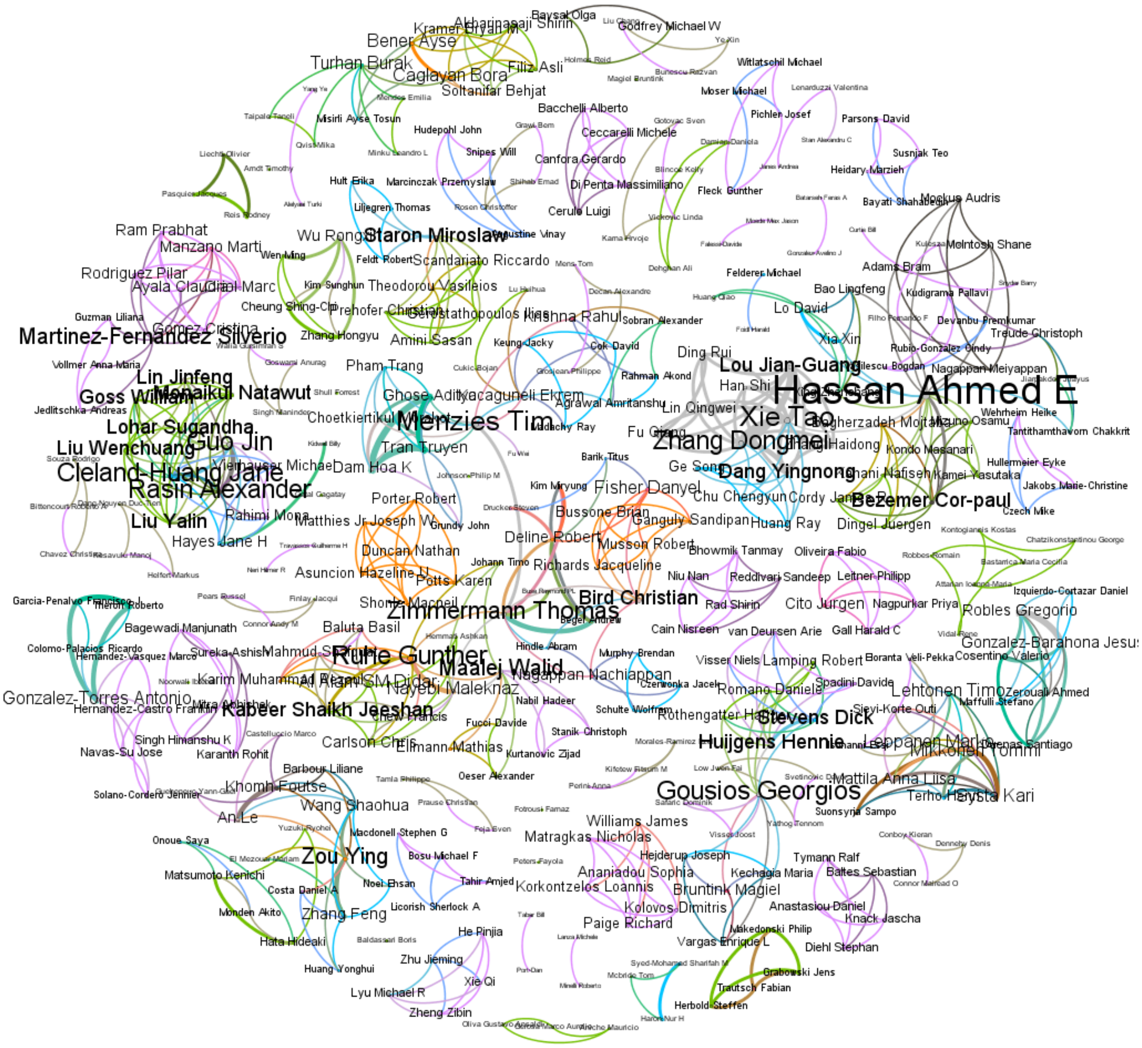
SOURCE: Prepared by the author.

The largest sub-graph, including 54 authors nodes represents almost 16% of the total number of authors. Connected nodes have distance 1. The most substantial distance between two author nodes is 6, while the average graph-distance between all pairs of nodes is 2.76. The graph density measures how close the network is to complete. A complete graph has all possible edges and density equal to 1. This metric was used to measure the collaboration between the authors of the papers selected in this study. A greater connection between authors can be verified when the density of the graph is closer to 1. In our graph, the co-authorship graph density is 0.011. This value is considered low due to the existence of numerous connected components (COSENTINO et al., 2017). That means that there is plenty of room for collaboration between authors on different themes in the area.

2.2.7 Typical issues addressed to software analytics (MQ5)

This section introduces the categorization of issues addressed to the area of *software analytics* according to research focus, which was organized according to contribution type.

Figure 2.5 - Coauthorship-network.



SOURCE: Prepared by the author.

Table 2.13 presents the categories of issues identified. The number of papers per category appears in brackets. The list of all issues addressed in each study is shown in Appendix B.

Table 2.13 - Typical issues addressed to software analytics per contribution type.

Contribution type	Issues categories
Data analytics - techniques, methods and tools	Data analytics approaches (7); Software failures (6); Data-driven requirements (5); Releases and code integration (5); Project management (5); Teamwork and collaboration (4).
Predictive modeling	Predictive models building (10); Defect and fault prediction (8); Effort estimation (4); Predictive modeling in practice (3); Predictive modeling performance (2); Project management (2).
Data Mining - method and tools	Mining software repositories studies (7); Diagnosis of crashing faults (5); Software traceability (3); Data collection (3); Bug fixing process (2); Developers' communication (2); Users' feedback (2); Software integration (1); Requirement review (1).
Monitoring - measure, metrics, and indicator	Continuous monitoring (4); Development targeted analytics (4); Project management (3); Continuous delivery (2); Defect prediction building (1); Quality assurance (1).
SA Issues & Concepts	Analytics for software maintenance (5); Data science and analytics (4); Software analytics value (3); Software quality (2).
Visual software analytics	Process improvement (4); Software evolution (4); Requirements management (2); Test and code visualization (2); Continuous deployment (1).
Software analytics platform	Code quality (2); Repository mining studies (2); Maintenance of mobile applications (1); Test analytics platform (1); Bug fixer recommendation (1); Continuous improvement (1).
Software analytics projects implementation	Software analytics in practice (1); Continuous delivery (1); Analytics from team metrics (1); Agile methods and DevOps (1).

SOURCE: Prepared by the author.

2.2.7.1 Data analytics - techniques, methods and tools

Data analytics approaches

Shull (2014) suggests goal-directed methods to help the analyst recognize which goals can be addressed, which metrics available will answer the questions of interests, and which datasets will lead to useful insights. Bruntink (2014) investigated the quality of the software evolution data available in a large-scale online index and analytics platform for open source projects. Gousios et al. (2016) introduced the concept of streaming *software analytics* and proposed a data analytics infrastructure which unifies the representation of historical and current data as streams and enables high-level aggregation and summarization using a common query.

Theodorou et al. (2017) introduced a reference architecture for data-driven systems that can change used data sources and data analysis algorithms at runtime to preserve the quality of data analytics. Cito et al. (2017) proposed context-based analytics that makes the links between runtime information and program-code fragments explicit by constructing a graph based on an application-context model. Compared to a standard diagnosis approach, context-based analytics decreases the number of required analysis steps, and the number of needed inspected traces. Ellmann et al. (2017) proposed an approach to categorize and identify development screencasts by conducting a similarity analysis of the transcripts and the Javadoc of the corresponding screencasts. Cosentino et al. (2018) presented an analytics tool called Graal, which was designed to conduct ad-hoc analyses of source code by supporting cross-cutting analysis on software project data.

Software failures

Lou et al. (2013) reported their experience about data-analysis techniques developed to solve engineers' pain points in incident management, where the massive data scale, sophisticated problem space, and incomplete knowledge were the main challenges faced by researchers. Kidwell e Hayes (2015) described the utility of fault classification in the context of *software analytics* and proposed an automated approach to learning a fault taxonomy by using machine learning techniques. Krishna (2017) proposed two algorithms (XTREE and BELLTREE) for generating a set of actionable recommendations on how to undertake code reorganization in order to reduce defects in code. Batarseh e Gonzalez (2018) introduced the analytics-driven testing (ADT), a method to predict software failures in the subsequent agile sprints and their locations with a specified statistical confidence level. This method uses a forecasting regression model for estimating where and what types of software system failures are likely to occur. Lou et al. (2017) reported their experiences about a set of data-analysis techniques developed and the Service Analysis Studio (SAS) system deployed to Microsoft' data centers for incident management. Bagherzadeh et al. (2018) analyzed the changes that were made to Linux system calls during the last decade by manually identify the type of changes and bug fixes that were made.

Data-driven requirements

Maalej et al. (2016b) introduced concepts on data-driven requirements engineering involving user feedback analytics along with usage data, logs, and interaction traces automatically collected.

Maalej et al. (2016a) carried out an experimental study to evaluate some probabilistic techniques (i.e., text classification, natural language processing, and sentiment analysis techniques) used to classify app reviews into four types: bug reports, feature requests, user experiences, and text ratings. From experiment results, they designed a review analytics tool for filtering critical reviews and assign them to the appropriate stakeholders. Fotrousi (2016) proposed quality-impact analytics of software products, features, and requirements based on a joint analysis of software quality and user feedback. Such analysis method should guide the verification and validation of functional and quality requirements as well as capturing new requirements. Hemmati et al. (2018) proposed a method for the collection, processing, and analysis of multiple data sources (i.e., requirements, bug reports, and usage data) to investigate high-level connections between user requirements and system utilization. Morales-Ramirez et al. (2018) proposed a technique based on speech-acts for the analysis of online discussions in order to discover relevant information that could enhance requirements elicitation.

Releases and code integration

Souza et al. (2015) investigated how rapid releases essential for the timely release of new versions affect code integration by analyzing how the backout rate evolved during Mozilla's process change. Rosen et al. (2015) proposed a data analytics tool for analysis and predictions on risky commits, called Commit Guru. This tool automatically identifies risky (i.e., bug-inducing) commits and builds a prediction model that assesses the likelihood of a recent commit introducing a bug in the future. Rahman et al. (2018) analyzed the practice of making frequent commits in open source software projects compared to proprietary projects, in order to investigate the expected benefits after continuous integration adoption regarding software improvement. Baltes et al. (2018) analyzed the commit and merge activity in GitHub projects that introduced the hosted continuous integration system. The aim was to investigate if commit activities were being adjusted towards the small incremental changes to software projects, after start using continuous integration. Nayebi et al. (2019) proposed an analytical approach called the Gandhi-Washington Method (GWM) to analyze the impact of recurring events in software projects, which uses regular expressions to condense and summarize information and infer treatments automatically.

Project management

Chatzikonstantinou et al. (2013) proposed a data analytics framework where analytics objectives are represented as goal models with conditional contributions. After the goal models are transformed into rules, they are assessed by a probabilistic reasoner using Markov Logic Network. Asuncion et al. (2013) presented a technique that uses change entries to obtain relevant project information, called FACTS PT (Flexible Artifact Change and Traceability Support for Project Team). This technique consists in automatically extracts, traces, aggregates, and visualizes change entries along with other software metrics to provide project information. Karim et al. (2016) applied various data analytics techniques to understand the factors responsible for estimation inaccuracy of project issues, verify whether the company guidelines regarding issues are really followed by practitioners, and investigate whether a prediction on estimation inaccuracy can support managers to take proactive measures. Guo et al. (2016) presented two data analytics methods to address the cold-start problem. The first identifies the best overall performing configuration, across all projects in the profile, and adopts that default configuration for use in cold-start projects. The last one matches the project characteristics of a cold-start project against those in the configuration profile, selects the most similar project, and then adopts that project's configuration for the cold-start project. Lin et al. (2017) designed a web-based tool that allows users to express software-related queries verbally or written in natural language. Those queries expressed in natural language are transformed into SQL and then executed against a centralized or distributed database.

Teamwork and collaboration

Gonzalez-Barahona et al. (2013) employed analytics techniques to study two open-source software development communities (WebKit and OpenStack). They conclude that such analytics can improve factual knowledge about how development communities are performing in aspects that are of interest to companies. Alelyani e Yang (2016) suggested a method to analyze worker behaviors when registering and carrying out the announced tasks within a software crowdsourcing context. Onoue et al. (2016) analyzed the characteristics of the population structures of OSS projects on GitHub using a population projection method which calculates the survival rate based on past population. Devanbu et al. (2017) used circular statistics to analyze the dispersion of timezones and work times, evaluating how timezone dispersion affects work hour dispersion.

2.2.7.2 Predictive modeling

Predictive models building

Based on the advances in the areas of data mining and predictive modeling, Menzies (2012) pointed out the need for evolution of prediction systems to decision systems and then to social reasoners. Kocaguneli et al. (2013a) analyzed trends in building prediction models when data need be transferred between different contexts and the information are scarce; and discussed synergies between different machine learning methods (transfer, semi-supervised and active learning) which may overcome these issues. Minku et al. (2016) discussed the role of software engineering experts when adopting data mining approaches and the lack of involvement of them in the process of building predictive models. Dam et al. (2016) introduced the deep model of software as an end-to-end generic framework based on deep learning for modeling software and its development process to predict future risks and recommend interventions. Zhu et al. (2017) proposed an approach to leverage historical usage data from users for constructing of context-aware reliability prediction models from web services.

Using different methods for textual similarity analysis, Nayebi et al. (2017) found that the combination of machine learning techniques with experts manually added labels has the highest prediction accuracy on change impact. Czech et al. (2017) used label ranking algorithms complemented with appropriate kernels to provide a similarity measure for predicting rankings of software verification tools. Menzies (2018) explored how *software analytics* can offer a somewhat accurate prediction about software attributes (e.g., cost or defect) of future projects although, in theory, often software project behavior is not predictable. Peters (2018) focused on finding a balance between privacy concerns and the utility of data for analytics, suggesting a combination of minimization and obfuscation to confidently share data with other projects and organizations to build prediction models. Menzies e Zimmermann (2018) provided an overview of how *software analytics* can help software engineers for discovering, verifying, and monitoring the factors that affect software development. In addition, they discussed a future where both qualitative and quantitative methods should be alternately used to explore software data; a model learned for one project may be applied to another, and complex models generated via *software analytics* need to be easily understood by humans.

Defect and fault prediction

Catal (2012) discussed challenges in building accurate fault prediction models to detect error-prone modules before the testing phase and pointed out that new software fault prediction tools should be developed and integrated with existing integrated development environments. With the aim to improve the efficiency of the testing process, Taipale et al. (2013) proposed a defect prediction model including different modes of information representation of the data and the model outcomes for guiding practitioners to focus their activities on the most problematic parts of the software. Soltanifar et al. (2016) suggested the use of code smells metrics along with churn metrics to train defect prediction models. Jiarpakdee et al. (2018) introduced an automated metric selection approach for interpreting defect models based on correlation analyses, using the Spearman rank test and the variance inflation factor analysis. An et al. (2018) investigated the characteristics of commits that lead to crashes into the Mozilla Firefox browser, and then they built predictive models to improve the process of detecting and fix the crash-prone code early when their developers commit code. Huang et al. (2018a) explored supervised and unsupervised models for effort-aware just-in-time defect prediction and proposed an improved supervised model called CBS+, which leverages the idea of both models. Agrawal et al. (2018) proposed an approach to combine data mining and optimization for *software analytics*, where data miners can generate the defect prediction models that are explored by optimizers, while optimizers can advise how to best adjust the control parameters of a data miner. Barbour et al. (2018) examined clone genealogies to identify fault-prone patterns of states and changes and found that adding clone genealogy information can increase the explanatory power of fault prediction models.

Effort estimation

Kocaguneli et al. (2013b) proposed an active learning algorithm for effort estimation by investigating how active learning changes the time required for making effort estimation models. Dehghan et al. (2016) presented a hybrid model for task completion effort estimation to be adaptable to a larger number of tasks. Choetkiertikul et al. (2018) proposed a prediction model for estimating story points based on two deep learning architectures (long short-term memory and recurrent highway network). Karna et al. (2019) produced an effort estimation model for software projects using a modified data mining approach where prior to model creation, additional clustering of projects is performed.

Predictive modeling in practice

Misirli et al. (2013) investigated practitioners' expectations of prediction models from three *software analytics* projects in order to understand how these models can be used in policymaking. The projects involved a defect prediction model, an early effort estimation prototype, and a quality prediction project. Tantithamthavorn e Hassan (2018) discussed some pitfalls and challenges observed in practice when practitioners attempt to develop analytical models for defects prediction. Dam et al. (2018) discussed explainability as a critical measure for developing *software analytics* prediction models based on social science, explainable artificial intelligence, and software engineering.

Predictive modeling performance

Focused on exploring the problem of transferring data from one project to another for the purposes of data analytics, Krishna e Menzies (2018) proposed the use of the bellwether method, where given N projects from a community one exemplary project is the project whose data yields the best predictions on all others. Kondo et al. (2019) analyzed the impact of eight feature reduction techniques on the performance and the variance in performance of five supervised learning and five unsupervised defect prediction models.

Project management

Choetkiertikul et al. (2015) used networked data classification to predict the degree of delay for a group of related tasks, providing automated support in predicting whether a subset of software tasks in a software project has a risk of being delayed. In order to provide an early warning as to whether questions will be raised by a developer in an issue report at the issue report filling time, Huang et al. (2018b) built a prediction model to capture issue reports when they are submitted.

2.2.7.3 Data mining - method and tools

Mining software repositories studies

Menzies (2013) compares and contrasts four kinds of data miners: algorithm miners that explore tuning parameters in data mining algorithms, landscape miners that reveal the shape of the decision space, decision miners that comment on how best to change a project, and the discussion miners that help the community debate trade-offs between the different decisions.

Robles e González-Barahona (2013) reported the experience of using mining software repositories techniques to detect plagiarism in a multimedia networks course where students have to submit several software programs. Williams et al. (2014) used an open-source telemetry platform to analyze ten modeling projects hosted by the Eclipse Foundation, mining the required data from the version control systems, bug tracking systems, and mailing lists. McIntosh et al. (2015) analyzed version histories from thousands of software repositories, three software ecosystems, and four large-scale projects in order to understand the prevalence of different build technologies and the relationship between build technology and build maintenance. Aniche et al. (2015) presented a heuristic based on static code analysis using cyclomatic complexity metric and the number of unit tests per method in order to statically calculate code coverage and make feasible for large-scale mining software repositories studies. Bayati et al. (2015) proposed a framework for mining and analysis of GitHub projects using features and metrics derived from historical data in repositories, object-oriented programming metrics, and the influences of developers on source codes. Low et al. (2015) sought out patterns using machine learning algorithms in projects hosted in GitHub in order to understand how free and open-source software projects survive.

Diagnosis of crashing faults

Wu (2014) described a set of techniques to assist the diagnosis of crashing faults, including a framework of collecting call sequence, a framework of crash bucketing, and a framework of locating crashing faults. Ye et al. (2014) proposed an approach for ranking source files of a project with respect to how likely they are to contain the cause of the bug. Their learning-to-rank technique uses domain knowledge to recommend relevant files for bug reports, making it easier to find the cause of the bug. Wu et al. (2014) mined crash reports, bug reports, and change logs using a technique for locating crashing faults based on crash stacks and static analysis techniques called CrashLocator. Wu et al. (2018) proposed a method to automatically locate crash-inducing changes for a given bucket of crash reports, useful to down the root causes and reduces the search space of bug fix location. An et al. (2019) investigated bugs that were caused by third-party DLL injections into the software ecosystems.

Software traceability

Tamla et al. (2017) proposed an architecture to implement an approach to automatically track critical changes to its origin from collecting meta information on a code example.

Hindle et al. (2015) presented a method for traceability and information retrieval to extract requirements topics and bug report topics by performing topic analysis on the documents and then inferring and linking these topics across all of the commit log messages in the source code repository. Bao et al. (2018) proposed a framework to improve the generalizability of the data collection, which uses operating-system-level instrumentation to track developer interactions. Also, they proposed an approach to segment and label the developers' low-level actions into a set of meaningful development activities by using machine learning.

Data collection

Finlay et al. (2014) described the extraction of metrics from a source control system using the application of data stream mining techniques to identify useful metrics for predicting build success or failure. Moser et al. (2015) used static code analysis to extract mathematical formula and decision tables from program statements and generate documentation from annotated source code. Dueñas et al. (2018) presented a free software tool called Perceval to perform automatic and incremental data gathering from the different data source - e.g., issue tracking systems, mailing lists, forums, source code repository, and social media.

Bug fixing process

Mezouar et al. (2018) analyzed the short messages posted by end-users on Twitter to investigate the usefulness of the social network in the bug fixing process. Noei et al. (2019) proposed an approach to map issue reports that are recorded in issue tracking systems to user-reviews in order to prioritize user-related issue reports of mobile applications.

Developers' communication

Cerulo et al. (2015) proposed an approach based on Hidden Markov Models for extracting the content of developers' communication which mixes different coding languages (e.g., source code, stack dumps, and log traces) with natural language parts. Fu e Menzies (2017) described a method to explore large programmer discussion forums, using a convolution neural network to predict whether two questions along with its entire set of answers posted on the forum are semantically related.

Users' feedback

Licorish et al. (2015) used data mining and natural language processing (NLP) techniques to investigate the issues that were logged by the Android community, and how Google's remedial efforts correlated with users' requests. Suonsyrjä et al. (2016) proposed a framework to support practitioners in finding a suitable technological approach for automated collecting of usage data within the process of data-driven software development.

Software integration

Yuzuki et al. (2015) presented techniques for identifying when conflicts occur in merging and detecting conflict resolution in method level. To understand how conflicts are resolved in practice, they analyzed ten open source projects written in Java.

Requirement review

Singh et al. (2017) developed an automated mining approach to validate requirement reviews using supervised learning classifiers and natural language processing over part of speech tags.

2.2.7.4 Monitoring - measure, metrics, and indicator

Continuous monitoring

Johnson (2013) discussed the trade-offs in designing analytics for software processes and products taking into account the degree of automation, the level of overhead developers and management incur to obtain the analytics, barrier to adoption incurred by the technique or technology, and a range of analytics that can be developed. Suonsyrjä e Mikkonen (2015) proposed an unobtrusive analytics framework for monitoring Java applications in order to help developers make informed decisions around the software project and the code review process. Barik et al. (2016) reported the experiences of the professionals at Microsoft during the transition to a data-driven culture, cataloging activities that leverage data at rest (logs) and streaming data (telemetry), identifying challenges in conducting these activities, and describing tensions that emerge as event data flow through these activities. Janes et al. (2017) proposed a continuous issue and error monitoring approach for small and medium enterprises to maintain quality above a minimum threshold supported by a recommendation system of quality practices and quality actions based on created errors.

Development targeted analytics

Treude et al. (2015) investigated how developers would measure the development activity, what information they would expect in a summary of development activity, and what factors influence how such an activity can be condensed into summaries (textual or numeric). Bruntink (2015) explored typical values to measure code size and growth (using lines of code) and observed large dispersion, skew and outlier rates for those metrics with the purpose of obtaining a base rate in *software analytics*. Cito (2016) investigated approaches to support developers in their decision-making by incorporating runtime information in source code and provide live feedback in IDEs by predicting the impact of code changes. Syed-Mohamad et al. (2017) proposed test-defect coverage analytic model to measure the test adequacy, which combines test and defect coverage information presented in a dashboard to help to decide whether the tests planned are enough.

Project management

Baldassari (2012) proposed an approach to software project quality measurement called SQuORE, with the purpose of retrieving information from several sources, compute a consolidation of the data, and show an optimized report on software or project state. Manzano et al. (2018) proposed an on-time delivery indicator in rapid software development to detect development problems in order to avoid delays and to estimate the additional time needed when specific requirements are considered. Decan et al. (2019) proposed metrics to analyze the growth, changeability, reusability, and fragility of the dependency networks into packaging ecosystems.

Continuous delivery

Huijgens et al. (2017) explored whether data mining techniques can help to define agile metrics with high predictive power for continuous delivery. Neri e Travassos (2018) investigated multidimensional relationships between software product characteristics to construct a continuous experimentation infrastructure to evaluate the software quality from continuous development environments.

Defect prediction building

Falessi e Moede (2018) proposed a desktop application called Pilot Defects Prediction Dataset Maker (PDPDM) to define the better set of product and process metrics to be used in the defect prediction model.

Quality assurance

Martínez-Fernández et al. (2018) proposed a quality model (called Q-Rapids quality model) composed with relevant metrics as well as product and process factors for actionable analytics in the rapid software development context.

2.2.7.5 Software analytics issues and concepts

Analytics for software maintenance

Dang et al. (2017) reported their experiences on transferring a code-clone detection and analysis approach to practice and their efforts to adapt the supporting tool to addresses the needs of real scenarios into projects with their inherent characteristics. Port e Taber (2018) reported examples of the use of a software metrics and analytics program for strategic maintenance of critical software, which actions depend on reliably knowing how many defects can be found, the likelihood of defects surfacing at a critical moment, and how much time it will take to find out and fix a defect. Ellmann (2017) discussed issues on the similarity of software documentation and the challenges of understanding the characteristics of similar software development documents. Godfrey (2015) discussed some of the dimensions of the problem of extracting and reasoning about the provenance of software development artifacts. Nguyen et al. (2018) discussed some challenges in discovering insights from the usage data from cloud-based applications.

Data science and analytics

Begel e Zimmermann (2014) investigated questions that software engineers would like data scientists to investigate software processes and practices, and then ranked the most important issues to work on first from the software engineers point of view at Microsoft. Kim et al. (2016) investigated the competencies and working styles of data scientists in software-oriented data analytics context. Arndt (2018) explored how big data techniques can be used to improve SE processes and how analytics methods allow for capturing events from large sets of data to turned into actionable intelligence for decision making. Kim et al. (2018) investigated the activities of data scientists at Microsoft clustering them based on the time spent in each activity.

Software analytics value

Robbes et al. (2013) explored how smaller companies would benefit from *software analytics* that has less manpower and less historical information in their repositories.

They observed that smaller companies can benefit from *software analytics* by breaking work down in a series of precise tasks that are assignable, estimable, and trackable. Conboy et al. (2018) examined business value in analytics based on the temporal complexity and identified a set of temporal factors that may affect the business value when people use analytics in different contexts. Baysal (2013) reported some experiences about extracting and analyzing data from software repositories to support decisions around the code review process, understanding user adoption of software systems, and improve developers' awareness of their working context.

Software quality

Foidl e Felderer (2016) discussed the challenges related to defect prevention and highlighted the importance of *software analytics* to improve quality assurance in the Internet of Things applications. Felderer (2016) discussed issues on descriptive, generating and predictive software quality models, such as the maintenance of models, traceability between quality models and unstructured artifacts, and integration of *software analytics* and runtime information, to name a few.

2.2.7.6 Visual software analytics

Process improvement

From an action research approach, Lehtonen et al. (2013) proposed along with company practitioners visualization the issue management system's data as a tool to identify problems in the agile development process and to make them visible for all stakeholders. Musson et al. (2013) reported experiences from *software analytics* project at Microsoft where visualizations help development teams identify and prioritize performance issues by focusing on performance early in the development cycle, allowing evaluating progress, identifying defects, and estimating timelines. Baysal et al. (2013a) argued about the importance of qualitative dashboards designed to improve developers' situational awareness by providing task tracking with custom views to help manage their workload while performing day-to-day development tasks. Mattila et al. (2017) reported an industrial case where data visualization of issue management system data was able to reveal deviations between planned process and executed process.

Software evolution

González-Torres et al. (2013a) proposed a framework for the visual *software analytics* process to understanding the software evolution with active participation of users.

Torres et al. (2013) presented the implementation of an architecture based on the evolutionary visual *software analytics* process and described how it supports knowledge discovery during software maintenance tasks. González-Torres et al. (2016) presented an application of visual analytics to software evolution developed to identify the software items that have been changed by a group of programmers, determining which programmer has led a project or contributed more to the development and maintenance in system reviews. González-Torres et al. (2018) proposed the design of an architecture for evolutionary visual *software analytics* which should retrieve the source code from different version systems, carry out the advanced analysis of programs written in different languages, and make visible the results of calculation of software metrics through visual representations.

Requirements management

Focusing on dynamic visualizations to support *software analytics*, Reddivari et al. (2014) proposed a framework for visual requirements analytics composed of five conceptual goals: user, data, model, visualization, and knowledge. The authors used the goal–question–metric to define the conceptual goals, as well as the set of questions for operationalizing the goals. Noorwali (2018) introduced the stakeholder concern-driven requirements analytics to fill the lack of readily available metrics and analytical methods that could aid the requirements management process.

Test and code visualization

Feldt et al. (2013) explored how visualization and correlation between test and code measurements can support decisions on software quality improvements, based on a case study where heatmaps were employed to visualize and monitor changes and identify recurring patterns of an embedded control system. Haron e Syed-Mohamad (2015) proposed a visual analytics tool to assess and validate the testing results using bubble charts to represent number of defects, branch coverage, and lines of codes per software component.

Continuous deployment

Mattila et al. (2015) demonstrated how to create an easy-to-interpret visualization mashing up software data from the issue management system, development data from the version control system, and actual end-user usage data.

2.2.7.7 Software analytics platform

Code quality

Czerwonka et al. (2013) presented a *software analytics* platform called CODEMINE build for collecting and analyzing data from Microsoft's products. The platform has been adopted by the product teams for data acquisition and analysis as part of a product development process, for customized analyses focusing on answering a specific question, or as a source of information and inspiration for new lines of research. Vargas et al. (2018) outlined a real-time *software analytics* platform named CodeFeedr to improve the quality and speed of decision making, monitor software development infrastructure in real-time, and create customized views of the workflow.

Repository mining studies

Focusing on replicability and validity of repositories mining studies, Trautsch et al. (2016) presented a smart data platform named Smart SHARK for data acquisition from several projects in order to create different analytic examples. The platform implemented combines automated data collection from different sources and Apache Spark to perform *software analytics* on the collected data. Trautsch et al. (2018) improved the design and the data collection process of the SmartSHARK, and added a table that shows the need for data storage for new plugins.

Maintenance of mobile applications

Minelli e Lanza (2013b) presented a web-based *software analytics* platform named SAMOA for structural analysis of mobile applications based on the source code, usage patterns, and historical data.

Test analytics platform

Liechti et al. (2017b) proposed the concept of test analytics that defined as "analytics on test-related data in order to give actionable insights about product quality and agile practices, with the goal to support a continuous improvement process". Then, they present a test analytics platform built to collects test-related data, analyze it, and give feedback to the team.

Bug fixer recommendation

Sureka et al. (2015) presented a decision support platform for guiding and assisting recommendations for the task of automatic bug assignment using textual information content from bug reports and non-textual features such as developer workload, experience, and collaboration network.

Continuous improvement

Liechti et al. (2017a) discussed the dimensions that need to be considered when introducing a data-driven continuous improvement process in agile organizations such as the nature of the channels through which the software and process metrics are published.

2.2.7.8 Software analytics projects implementation

Software analytics in practice

Zhang et al. (2013) reported lessons learned about the *software analytics* system produced for Microsoft product teams. The lessons reported include focusing on problems that practitioners care about, using domain knowledge for correct data understanding and problem modeling; building prototypes early to get practitioners' feedback, taking into account scalability and customizability; and evaluating analysis results using criteria related to real tasks.

Continuous delivery

Huijgens et al. (2018) examined what factors helped and hindered the implementation of *software analytics* in an environment of continuous delivery. Main success factors refer to prior defining and communicating the aims, standardization of data, build efficient visualizations, and use of an empirical approach.

Analytics from team metrics

Augustine et al. (2018) reported lessons learned from deploying a *software analytics* solution focused on metrics and indicators around the improvement of agile teams from a multinational organization.

Agile methods and DevOps

Snyder e Curtis (2018) reported how *software analytics* were used to guide improvements and evaluate progress during an Agile/DevOps transformation in a software company, including challenges in selecting measures and implementing analytics.

2.3 Findings

In this section, we discuss the findings and research gaps based on mapping results. In general, when assessing the quality of the selected studies, we found that research in the field of *software analytics* has been of high relevance to industry. Few studies were conducted in an academic setting involving students or using toy examples. On the other hand, many studies deal with real cases from the industry, but few report involvement with professionals. The number of case studies and experiments in industrial contexts is quite significant. The rigor scoring should enable locating studies described in a way that facilitates replication, reproducing, and synthesis of evidence (IVARSSON; GORSCHEK, 2011). However, studies reporting lessons learned tend to be less rigorous concerning study design and threats to validity.

Noticeably, research related to *software analytics* has been increasing year by year and becoming more robust, given the number of studies published in journals over the past year. Over the years, many solutions in *software analytics* have been implemented and evaluated in practice, so that few studies refer to validations performed in academic settings. *Software analytics* research is closely associated with problem-solving and process improvement, having practical value to the software industry. By observing the co-authorship network, we found that there is still plenty of room for collaboration with both academic researchers and industry professionals.

Many studies have contributed techniques, methods, and tools to support data analysis focused on more specific issues target software requirements, fault detection, and project management; but also with general approaches that take into account data type and information flow. *Software analytics* has a broad spectrum of activity that permeates the areas of predictive modeling, maintenance, and software evolution. Several studies proposed models for defect and fault prediction, while others concern on models building, focusing on the evolution of predictive technologies. Means to increase the degree to which a practitioner observer can understand the reasons behind a specific prediction is an open question and interest topic in this domain.

Mining software repository studies provide methods and better ways of discovering useful information on different software issues. However, there are few tools for automation of data gathering from different data sources, coming out various forms (both structured and unstructured). Natural language processing and text mining techniques have been applied to automatically analyze users' feedback and to validate requirement reviews, for instance. But there are still few contributions in these areas.

Visual *software analytics* is commonly employed in software process improvement and software evolution. We found few studies using visual *software analytics* for supporting the decision making in issues target testing process and software quality assurance. In the domain of data-driven software development, data science knowledge is critical to power *software analytics* process. However, an issue to be investigated is whether the lack of in-depth knowledge and expertise in data science can make a *software analytics* project infeasible in small teams.

Concerning monitoring activities, researchers and practitioners have been investigating how to measure issues related to development activity, software quality, and project management, especially within continuous development environments. Proposals of analytics software platforms have been increasing over the past two years, but there is little evidence about the practical value of these platforms and challenges that practitioners face when using them. We found that continuous experimentation and DevOps culture are emergent themes that can leverage the implementation of *software analytics* projects. By the way, we found a small number of studies on the implementation of *software analytics* projects in practice.

2.4 Threats to validity

The threats of this research mainly include issues related to the reliability of primary study selection, inaccuracy in data extraction, potential researcher bias, and generalizability, according to Zhou et al. (2016).

Reliability of primary study selection: As for the selection of studies, we conducted the search in two stages, combining the method of automatic search in multiple databases (seven digital libraries well-known in the software engineering field) and snowballing forward method from five key papers in the *software analytics* area. In the second stage, we included a substantial number of relevant studies (44 articles) which represented 31% of total selected studies.

In order to reduce the threat of exclusion of relevant papers during the screening process, only papers that are not in the area of software engineering, or clearly did not address issues related to *software analytics*, were excluded. Moreover, the reasons for the exclusion decision were documented. The selected papers were twice checked to detect and remove duplicate papers. Finally, to mitigate inappropriate selection of papers, some inclusion and exclusion criteria were taken into account to ensure the inclusion of only relevant articles.

Inaccuracy in data extraction: To avoid bias in data extraction, we established a data extraction form to capture the information required for answering our research questions. However, only a single researcher performed data extraction. To mitigate this threat, (i) the data extraction process was twice checked, (ii) an existing classification was used to categorize the types of research, and (iii) the analysis of keywords together with title, and abstract were used to categorize the domain, contribution type, and the addressed *software analytics* issues.

Potential researcher bias: An incorrect classification and unsatisfactory data synthesis could be caused by subjective researcher interpretation about the extracted data. As regards these issues, there is a potential threat to the validity of this study, since a single researcher conducted these activities. To reduce this threat, the review protocol, quality assessment criteria, data extract form, and results were reviewed by the research project supervisors. The double-checking was adopted by the researcher as a method to check the reliability of searches, the inclusion and exclusion criteria, and the studies classification accuracy.

Primary study generalizability: Considering the number of selected primary studies and the time span, our data extraction and classification can be representative of different domains. In this study, we identified the domains to which a study's findings can be generalized.

2.5 Chapter summary

This chapter presented a systematic mapping that provided us an overview of the SA area. By categorizing the main issues addressed to *software analytics*, we were able to identify the SA areas that have received the least attention.For instance, the results of this review indicate that there are few studies related to the implementation of *software analytics* projects in practice (ZHANG et al., 2013) (HUIJGENS et al., 2018) (AUGUSTINE et al., 2018) (SNYDER; CURTIS, 2018). This gap is explored in this thesis.

3 RESEARCH METHODOLOGY

This chapter presents the research methodology and design adopted in this study. The research methods, the techniques of data collection, and theoretical underpinnings used for data analysis are also presented. The application of these methods and techniques are reported in Chapters 6 and 7.

Section 3.1 describes the central approach used as a research framework for conducting the study. Section 3.2 describes the inquiry techniques used in data gathering and analysis during proposal evaluation. Section 3.3 outlines the activities and research design.

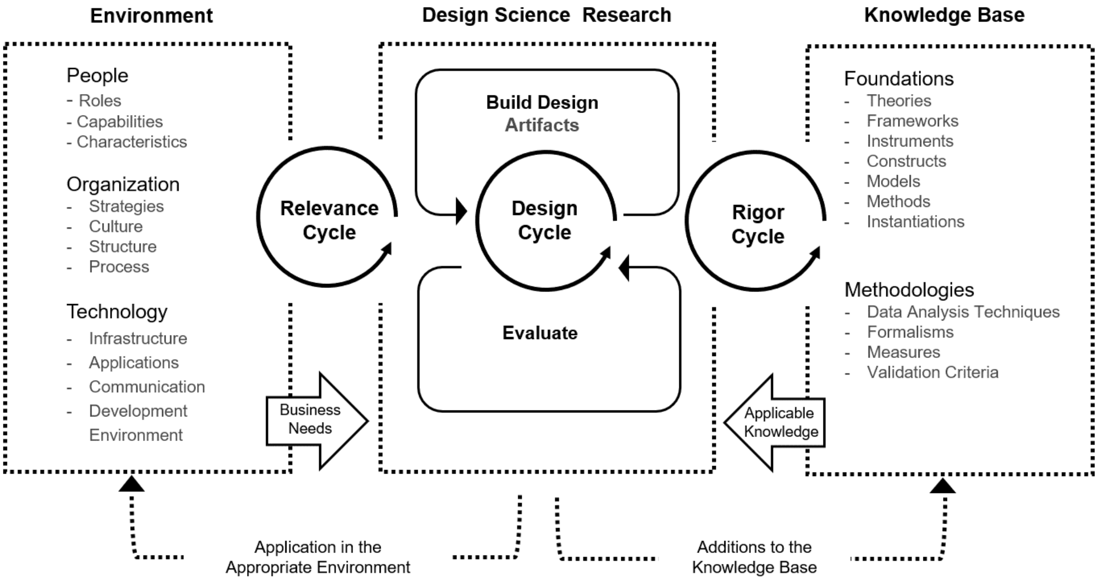
3.1 Research framework

The proposal of this research is an approach to supporting agile teams in *software analytics* activities. That is, we can consider that the ultimate goal of this work involves generating knowledge through making by designing artifacts seeking to extend the boundaries of human and organizational capabilities (PURAO, 2002). To accomplish this goal, Design Science Research (DSR) is adopted as the central research methodology. Centered on discovery-through-design, the main DSR focus is on problem-solving by means of the design of innovative artifacts to meet practical needs identified from a given domain (BASKERVILLE, 2008). The term "artifact" used herein is broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems).

DSR has its roots in the sciences and engineering of the artificial (SIMON, 1996). In the last 30 years, DSR has been widely adopted by the Information System community and associated areas, such as the Software Engineering and Computer Science. In these areas, DSR is commonly used to producing new ideas in order to improve the ability of people and organizations to adapt and succeed in environments that are constantly evolving in terms of the scientific and technological paradigms (GREGOR; HEVNER, 2013).

In this thesis, the DSR framework for building and evaluating of artifacts proposed by Hevner et al. (2004) is applied. Figure 3.1 presents the key components of Hevner et al.'s framework.

Figure 3.1 - DSR framework.



SOURCE: Adapted from Hevner et al. (2004).

According to Hevner e Chatterjee (2010), DSR is a problem-solving paradigm, where "knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact" (HEVNER; CHATTERJEE, 2010). As displayed in Figure 3.1, the *relevance cycle* to the right of the framework represents the inputs (requirements) for the research development. by identifying potential opportunities and problems in a real environment. The environment setting includes the people (roles, capabilities, and characteristics), organizational systems (strategies, structure, culture, and process), and technology available (infrastructure, applications, development environments, communication aspects). The relevance cycle also comprehends the definition of criteria to evaluate the research contributions.

At the left of the framework, the *rigor cycle* provides foundations and methodologies from a knowledge base. The examples of foundations are theories, frameworks, instruments, constructs, models, methods, and instances used in the developing and building phase of a research study. And, methodologies typically encompass data collection, empirical analysis techniques, and also provide guidelines to justify and evaluate the solutions. The new knowledge generated by the research will be added to the knowledge base later.

At the heart of the framework, the *design cycle* represents the construction and evaluation of an artifact guided on relevance and rigor. In an iterative way, the solutions are refined through feedback. The outcomes of this process are contributions to both research and practice. The design artifact is the most typical contribution of DSR. The evaluation rigor refers to using appropriate methods available in the knowledge base.

The research design of this thesis is based on the following guidelines proposed by Hevner et al. (2004):

- **Design as an artifact.** DSR must produce a viable artifact in the form of a construct, model, method or an instantiation.
- **Problem Relevance.** Objective of DSR is to develop technology-based solutions to important and relevant business problems.
- **Design Evaluation.** The utility, quality and efficacy of a design artifact must be rigorously demonstrated with well-executed evaluation methods.
- **Research Contribution.** Effective DSR must provide clear and verifiable contributions in the areas of the design artifact, design foundations and/or design methodologies.
- **Research Rigor.** DSR relies upon the application of rigorous methods in both the construction and evaluation of design artifacts.
- **Design as a search process.** The search for an effective artifact requires utilising available means to reach desired ends while satisfying laws in the problem environment.
- **Communication of research.** DSR must be presented effectively both to technology-orientated as well as management-orientated audiences.

3.2 Research methods

This section presents the methods applied in the DSR evaluation cycle divided into data gathering and analysis methods.

3.2.1 Data gathering

For data gathering, an observational study was applied to achieve a deeper understanding of the phenomenon studied.

Within software engineering, the realization of observational studies can lead to a better understanding of practices, and also to the elaboration of validated practice (D'ASTOUS; ROBILLARD, 2002). For example, observations can be conducted in order to investigate how software engineers conduct a certain task (WOHLIN et al., 2012). There are different types of observational methods. The case study and ethnographic studies are examples of observational methods because involve studying the spontaneous behavior of participants in natural surroundings. While a case study focuses on a phenomenon or particular case in the real-world context looking for a link between the natural setting and a phenomenon (RUNESON et al., 2012), the ethnography studies examine the details of a person, an organization, or a culture from both a broad perspective to investigate how a phenomenon be aroused and developed (AKTINSON; HAMMERSLEY, 1998). Structured observation is another type of observational method typically carried out in a laboratory rather than in a natural setting. In the structured observations, the researcher uses a standardized procedure and decides where the observation will take place and with which participants (PRICE et al., 2015).

Observation studies can be used in combination with a variety of techniques to collect data such as interviews, surveys, and analysis of documents and artifacts (BAKER, 2006). The data collected in observational studies are often qualitative in nature but they may also be quantitative or both (mixed-methods) (PRICE et al., 2015). Participants may or may not be aware of the research objective. Researchers can conduct observational studies on a participant or non-participant basis. In the participant observation, the researcher becomes more involved with the activities of the study participants, while in the non-participant observation, the researcher observes the study participants, with their knowledge, but without taking an active part in the situation under investigation (LIU et al., 2010).

The main limitations of observational studies are related to the observation effect and observer bias (LIU et al., 2010). The observation effect refers to the subject's awareness of being under observation. That is, the presence of the researcher (even not participating) may influence the participants' actions. This effect may be mitigated by a longer period of observation. Observer biases may be mitigated by ensuring systematic and rigorous approaches to sampling. To increase reliability, the observer can take steps through the collection of detailed field notes, and recording of audio and video.

In this study, a structured observational study was carried out to evaluate the use of the artifact proposed to support *software analytics* activities in a lab setting. To complement the observational study, the participatory design technique (PD) (BRATTETEIG et al., 2012) was employed to gather design suggestions from study participants. PD is usually used for engaging users in the design of new information technology (BRATTETEIG et al., 2012). There are different ways for supporting how PD can be done in practice (BODKER et al., 1995). Workshops and design sessions are the main methods of participatory design in which users by taking an active part in the activities and design decisions. During PD sessions users are encouraged to think creatively and propose their own ideas by assessment of sketches, prototypes, and mockups created by designers.

3.2.2 Data analysis

For analysis of observational data captured from interaction of the study's participants with the artifact proposed in the first design cycle, two approaches were adopted: a resources model (RM) (WRIGHT et al., 2000) and sequential analysis (SANDERSON; FISHER, 1994). RM is a conceptual framework based on Distributed Cognition theory (DCog) that allowed us to classify the interaction type between people and available resources used as activity support under observation. RM was chosen by considering that *software analytics* planning and managing is essentially characterized by cognitive activities that involve memory, planning, reasoning, inferring, learning, and making decisions (ROBILLARD et al., 1998), (HOLLAN et al., 2000). The sequential analysis, in turn, allowed us to explore the interaction patterns by observing the sequence of interactions and the most significant participant's actions over time. It is worth mentioning that the combination of these two approaches is an original contribution of this work. As far as we know, the RM and sequential analysis have never been used in combination. Our method of analysis using both approaches is described in detail in Section 6.2 from Chapter 6. Next, we introduce the concepts of DCog, RM, and sequential analysis.

3.2.2.1 Distributed cognition

Rooted in the cognitive sciences, Distributed Cognition (DCog) (HUTCHINS, 1995) has provided a sound theoretical basis for investigating cognitive activities (FURNISS; BLANDFORD, 2006), (WRIGHT et al., 2000). DCog is commonly used to explore how information acquisition and propagation occur by observing cognition as a process distributed across individuals, artifacts, internal and external representations, and space and time (HUTCHINS, 1995).

DCog extends the cognition study to beyond the individuals' brains by emphasizing a holistic and systemic view (HOLLAN et al., 2000). Hutchins and colleagues' works investigated the distributed cognition in airline cockpits (HUTCHINS, 1995) and collaborative programming activities during adaptive software maintenance (FLOR; HUTCHINS, 1992). In the ASD context, DCog has been adopted to understand cognitive activities characterized by remote working and collaboration in agile teams (SHARP; ROBINSON, 2006) (DESHPANDE et al., 2016) (SHARP et al., 2012). DCog is not a ready approach that one can easily apply in practice. A lot of time can be spent to understanding all concepts behind DCog theory.

3.2.2.2 Resources model

Wright et al. (2000) proposed the Resources Model (RM) grounded on DCog to identify unnecessary cognitive complexities in user interfaces. The authors use the term "*resource*" to mean a source of information used to achieve a given task taking into account external artifacts and a set of *abstract information structures* which can be distributed between people and technological artifacts. The RM identifies six *abstract information structures*, as follow:

- *Plan*: refers to a sequence of actions that could be carried out. Plans can be represented internally as memorized procedures to complete some task. They can also be represented externally as a step-by-step procedure, for example, checklists, or standard operating procedures, or user instructions. A plan prescribes an order in which actions should be carried out.

- *Goal*: specifies a state of the world to be achieved. Goal are abstract information structures that can be represented internally or externally.

- *Possibilities*: refers to the set of possible next actions that can be taken, given the current state of the system. The artifact or situation affords a set of possible actions.

- *History*: consists of a list of the actions already taken or states achieved, to get to the current state.

- *Action-Effect*: involves a set of possible actions are known; a statement of the effect that an action will have if it is carried out.

- *State*: refers to the collection of relevant values of the objects that feature in the interaction at a given point in the interaction.

Abstract Information structures may be represented internally in the head of the person, externally in the resource, or even distributed across the two. These structures can be combined as resources for action in different ways. To characterize the different configurations of resources, RM prescribes four *interaction strategies* as follows:

a) *Plan Construction* (PC): it involves to compare the current state with some goal state and to select from possible actions those that reduce the difference between the two states. In this strategy the resources required are: goal, possibilities, action-effect, and state.

b) *Plan Following* (PF): it involves the coordination of a predefined plan based on the history of the actions, the current state of activities undertaken, and a goal to follow. In this strategy the resources required are: : plan, history, state, and goal.

c) *Goal Matching* (GM): it is one alternative where decisions about what to do are more localized by matching the effects of action with the current goal and checking to see if the resulting state satisfies the goal. In this strategy the resources required are: goal, possibilities, action-effect.

d) *History-based Choice* (HC): it can be considered a starting point to reach a goal when external representations provide an interaction history to be used as a possibility. In this strategy the resources required are: goal, possibilities, history.

Wright et al. (2000) emphasized that a strategy does not emerge as a consequence of available resource configurations, but is an active process of interpreting external representations as resources. Scaife e Rogers (1996) highlight that the understanding of how external representations impact in the cognitive system can help in the future improvements of the users' work.

RM has been used for the representation of information structures as abstract concepts but also as an analytical tool to investigate how resource coordination occurs for action. Vick et al. (2003) investigated how distributed virtual teams engage in synchronous problem solving using decision modeling software supposing that analysis of patterns of time, and cognitive information trace use and re-use during the work process can be used to evaluate how effectively a team manages available resources under a variety of circumstances.

Fleury et al. (2015) used the RM to ascertain how automated assistance helps users to correct errors of architectural floor plans. Dubochet (2009) used the distinction between abstract information structures of the RM to discuss how knowledge was shared in a team of programmers. Smith et al. (1999) applied the RM to identify external resources needs to support in virtual environments by interaction episodes analysis.

3.2.2.3 Sequential analysis

Sequential analysis (SA) is an approach widely used in observational studies to explore interactive behavior when people use computing and communication technologies (OLSON et al., 1994). SA is based on observational data categorized according to some predefined set of codes to locate and describe patterns of interdependence that occur within a behavior stream (BAKEMAN; GOTTMAN, 1997). SA provides information about how patterns of behavior characterized by a sequence of events occur over time.

SA analysis is quite common in HCI research for the investigation of sequences of user interactions involving complex tasks (SANDERSON; FISHER, 1994). However, SA has been applied in SE research to discover patterns of behaviors from interaction events sequence involving individual or group work (D'ASTOUS; ROBILLARD, 2002) (POHL et al., 2016). Among various existing exploratory sequential data analysis method, Lag Sequential Analysis (LSA) (BAKEMAN; GOTTMAN, 1997) is a statistical method widely used to discover probabilistic patterns in the stream of events and indicate the degree of confidence with which we can state that a given event influences the occurrence of other.

d'Astous e Robillard (2002) applied a LSA to understand the collaborative activities performed during peer review meetings. First, they decomposed reviewers' verbal interaction into sequences representing a series of successive moves corresponding to a common subject, and then applied LSA to find significant relationships and exchange patterns that may occur during such meetings.

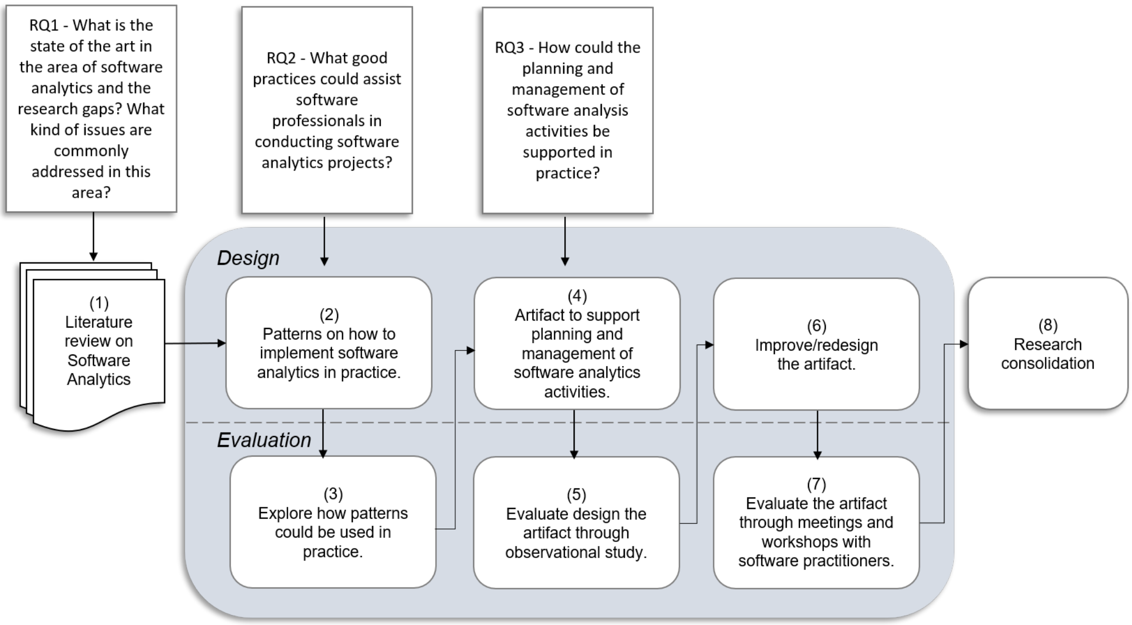
Pohl et al. (2016) used LSA for understanding the reasoning processes of users while interacting with software visualization artifacts. They gathered the interaction streams through log files and then used an existing information visualization taxonomy to categorized the user activities. Due to the emphasis on human reasoning and interaction of users with software visualization artifacts, their findings were also based on DCog theory.

Recently, Jeong (2019) used LSA to identify possible associations between instructional events observed in educational audio recordings (podcasts) considering different learner satisfaction levels. To coding the sequence of educational events, Jeong (2019) used a set of instructional events prescribed in a pre-established instructional model.

3.3 Research design

In this section, research design with the steps of conduction are presented. Eight activities were delineate to answer the three research questions, as displayed in Figure 3.2.

Figure 3.2 - Research design.


RQ1 - What is the state of the art in the area of software analytics and the research gaps? What kind of issues are commonly addressed in this area?
RQ2 - What good practices could assist software professionals in conducting software analytics projects?
RQ3 - How could the planning and management of software analysis activities be supported in practice?
Design
Evaluation
(1) Literature review on Software Analytics
(2) Patterns on how to implement software analytics in practice.
(3) Explore how patterns could be used in practice.
(4) Artifact to support planning and management of software analytics activities.
(5) Evaluate design the artifact through observational study.
(6) Improve/redesign the artifact.
(7) Evaluate the artifact through meetings and workshops with software practitioners.
(8) Research consolidation

SOURCE: Prepared by the author.

The first activity (1) refers to a literature review on the existing studies regarding *software analytics* to have a broad overview of the research area and provide insights for the proposal outlining. The second activity (2) includes identifying patterns to support software practitioners in *software analytics* activities. Then, a case study is conducted to explore how patterns could be introduced in practice (3). The forth activity (4) is the artifact design taking into account previous activities findings.

An observational study is conducted (5) to evaluate the artifact design in supporting to the planning and manage of *software analytics* activities. The sixth activity (6) is to refine the artifact design, and then a second round of evaluations is performed through workshops with software practitioners (7). Finally, the eighth activity (8) is the research consolidation.

3.4 Chapter summary

This chapter describes the DSR framework adopted as the central methodology of this research that allows the application of different design and evaluation methods. Then, the techniques we selected to collect and analyze data are presented. In particular, we describe the two approaches that were used for the analysis of observational data: the resources model (RM) (WRIGHT et al., 2000) and sequential analysis (SANDERSON; FISHER, 1994). Details on how these approaches were applied in the evaluation of the artifact proposed in this work are provided in Chapter 6. Finally, we explain the steps of the research.

4 A PATTERN LANGUAGE FOR SOFTWARE ANALYTICS

This chapter introduces a set of patterns for *software analytics* based on experience reports identified from the literature review. These patterns were discussed previously at Conferences on Pattern Languages of Programs (CHOMA et al., 2017) (CHOMA et al., 2018), and then documented in a pattern language format. By answering what good practices could assist software professionals in conducting *software analytics* (RQ2), the proposed pattern language describe steps to integrate the analytics activities into the development process. In the next chapter, the ideas behind the patterns will be used as the basis for SA Canvas design, an artifact to support the planning and management of *software analytics* activities in practice.

The remainder of this chapter is structured as follows: Section 4.1 describes the main concepts related to patterns and pattern language. Section 4.2 presents some patterns related to *software analytics* area. Section 4.3 contains a brief description of each pattern for *software analytics* and an overview of them and their relationships. And, Section 4.4 describes the patterns in more detail.

4.1 Patterns and pattern languages

The original concept of patterns emerged in the 1970s and was conceived by the architect Christopher Alexander (ALEXANDER, 1979). In his book "The Timeless Way of Building", Alexander (1979) presents the following definition of what a pattern is:

> The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must create it. It is both a process and a thing; both a description of a thing which is alive and a description of the process which will generate that thing (ALEXANDER, 1979).

Patterns in Software Engineering appeared in the 1990s when an influential collection of patterns for object-oriented software design was published by Gamma (1995). Nowadays, there are patterns for many domains and interests, such as analysis patterns, system test patterns, user interaction patterns, organization and process patterns. The annual Pattern Languages of Programming (PLoP)[1] conferences have become a permanent forum to discuss recurring problems of software design and other issues in such domains.

[1] http://www.hillside.net

In summary, patterns have been designed to capture successful solutions to recurring problems by documenting experiences. Basically, each pattern presents the context for the problem, the forces that weigh upon the problem-solver and suggests a proven solution to it (RISING, 1999). Patterns can be written in various forms, but a very usual way is the Christopher Alexander style (ALEXANDER, 1977). In general, a pattern writing is composed by the following elements:

- Name: a short description or a single word that is significant.
- Context: where the pattern might apply.
- Problem: what problem could be solved with the pattern.
- Forces: considerations on constraints/limitations conflicting with the goals.
- Solution: what to do to solve problem, how to achieve the goal.
- Examples: one or more applications using the pattern.
- Consequences: positive and negative effects of applying the pattern.
- Known uses: it describes known uses that confirm the pattern recurrence.
- Related patterns: patterns that are dealing with correlated issues.

Patterns can be applied in isolation. Nonetheless, patterns are commonly applied together from a language to solve a given problem. The group of interconnected patterns that fit together and fully address a topic or specific domain is named *pattern language* (BUSCHMANN et al., 2007).

Just as language is the main method of human communication, consisting of words used in a structure to put words together in meaningful ways, a pattern language is a language that comprises patterns and the rules to put patterns together also in meaningful ways (COPLIEN; HARRISON, 2005). In a pattern language, the patterns are ordered, connected, and presented in a certain sequence. Pattern languages are about emergent behavior in systems, while individual patterns encapsulate related forces that focus on specific trade-offs to guide the decisions (COPLIEN; SCHMIDT, 1995). Several pattern languages have been proposed in the engineering field to address different matters from organizational aspects (ITO et al., 2019) to more technical aspects such as the architectural and design patterns (GUERRA et al., 2013).

While, Ito et al. (2019) proposed a pattern language involving problems caused by the transfer of knowledge and responsibilities in the software industry during the transition of people to other parts of the company or when they retire, Guerra et al. (2013) proposed a pattern language involving software architectural patterns by defining a reference architecture which identifies a base structure for metadata-based frameworks.

4.2 Patterns in software analytics area

As far as we know, no previous study has presented patterns addressing the implementation of *software analytics*, nor related patterns for similar practices in agile teams. However, we found some patterns related to the analysis of specific software issues (MCGRATH et al., 2013) (GIGER; GALL, 2013) (SOUZA et al., 2013).

McGrath et al. (2013) identified a pattern to trace code changes from user requests to change implementation by analyzing mailing lists and code repositories called Concept to Commit. First, they suggest how to reduce the volume of data, and then how to analyze both emails and commits descriptions using basic text mining by performing the steps: tokenization, stemming, and document matrix creation. For this activity, they indicated some tools such as RapidMiner[2] or any statistical software program like R software[3]. Finally, frequency analysis can be performed using word cloud, heat map, or a dendrogram chart. This pattern suggests solutions to a very common problem within the context of *software analytics* that involves the analysis of unstructured data.

Giger e Gall (2013) presented a pattern called Effect Size Analysis related to significance testing used to determine whether the collected data support or not the researchers' hypothesis. They describe how significance testing can be extended by an analysis of the magnitude. This pattern is indicated for researchers want to draw more general conclusions and valid results using a restricted data subset. By addressing issues in the context of statistical decision making, researchers could rethink costly actions in response to such a comparatively small effect size.

Souza et al. (2013) identified two patterns related to cleaning up invalid bug data – Look Out for Mass Updates and Old Wine tastes Better. They refer to best practices to deal with missing or inaccurate data within bug tracking systems.

[2] Source: https://rapidminer.com/

[3] Source: https://www.r-project.org/

The first pattern is indicated to determine which changes to bug reports were the result of a mass update, while the second one is to determine bug reports that are too recent to be classified.

Baysal et al. (2013c) introduced a pattern called ARTIFACT LIFECYCLE MODEL to facilitate the analysis of software artifacts and its evolution throughout development. Such models are used to capture the dynamic nature of how certain development artifacts changed over time. For example, the status of the tasks, modifications to lines of code, or bugs fixing status.

As mentioned earlier, these patterns describe solutions to specific problems in the area of *software analytics*. However, the data analysis is only part of the process that aims at generating insights and supporting decisions. Besides analysis, *software analytics* process also comprises gathering, measuring and monitoring, and visualizing information (BUSE; ZIMMERMANN, 2012). It is further noted that the mentioned patterns address data analysis from a more technological perspective. In contrast, the patterns documented in our study focus on the human perspective, as regards the decision-making process.

4.3 Pattern language summary

This section introduces the proposed Pattern Language for Software Analytics whose purpose is to present viable solutions on how to implement *software analytics* activities in an agile software development contexts. As mentioned earlier, the patterns emerged from a literature review carried out in *software analytics* area. In this review, we searched for best practices in several experience reports and identified the typical issues addressed with *software analytics*.

The set of patterns that compose our pattern language were previously submitted in two PLoP conferences (CHOMA et al., 2017) (CHOMA et al., 2018). These conferences are venues for pattern development and dissemination, such as process patterns, analysis patterns, organizational patterns, security patterns, and architectural patterns (BUSCHMANN et al., 2007). PLoP conferences consist of two phases, *shepherding* and *writers' workshops* [4]. *Shepherding* occurs before the conference. In this phase, the authors receive suggestions for improvement of the paper from an experienced author – a shepherd. At the end of the shepherding process, the shepherd recommends whether to accept the submission for review at the conference.

[4] http://www.hillside.net/plop

The *writers' workshop* process occurs during the conference meeting. In these workshops, the participants are all authors who have submitted papers they want feedback on for improvement. The purpose is to review and help the author improve the paper. After the revisions and publishes in the Plops conferences, the patterns presented in this chapter were consolidated into a pattern language and submitted to the Journal LNCS Transactions on Pattern Languages of Programming.

Next, we present a summary of the eight patterns containing a brief description of each of them.

(1) What You Need to Know: To solve the issues that the team want to improve in the system and/or the software development process, in a context where there is a large amount of software data that can inform the decisions of the team, the solution is to define the key issues that the development team wants to focus on, in order to improve the software throughout the project.

(2) Choose the Means: To solve how to gather useful data regarding the issues that the team need to solve, in a context where a plethora of data is available, the solution is to define the most appropriate means, such as metrics, tools, techniques and other approaches for extracting data from software artifacts that will be useful in future decisions.

(3) Plan Analytics Implementation: To solve how to implement the *software analytics* activities fitting them to project roadmap along with other development tasks, in the context where the tasks directly related to the implementation of software features are the top priority, the solution is add tasks related to the *software analytics* in the backlog to be prioritized with the regular project tasks.

(4) Small Steps for Analytics: To solve how to implement *software analytics* in a pace that it does not to overburden the team, in the context where much information at the same time can confuse and make the team lose focus, the solution is to adjust *software analytics* tasks within the team schedule by breaking down them at smaller portions to be carried out in multi-steps.

(5) Reachable Goals: To solve how to turn *software analytics* findings into actionable insights to improve software aspects, in a context where to perform all improvements based on the analytics automated feedback might lead the team to act without focus, the solution is to take actionable insights from the *software analytics* findings, and from them, settle reachable goal adjusting the action steps.

(6) Learning from Experiments: To solve how to obtain information to make informed decisions about software issues on some aspect we have not yet implemented or we need to redesign, in a context where the team has nowhere yet to collect and analyze data to support their decisions, the solution is to create an alternative solution and perform an experiment collecting data that allow the comparison with the current solution.

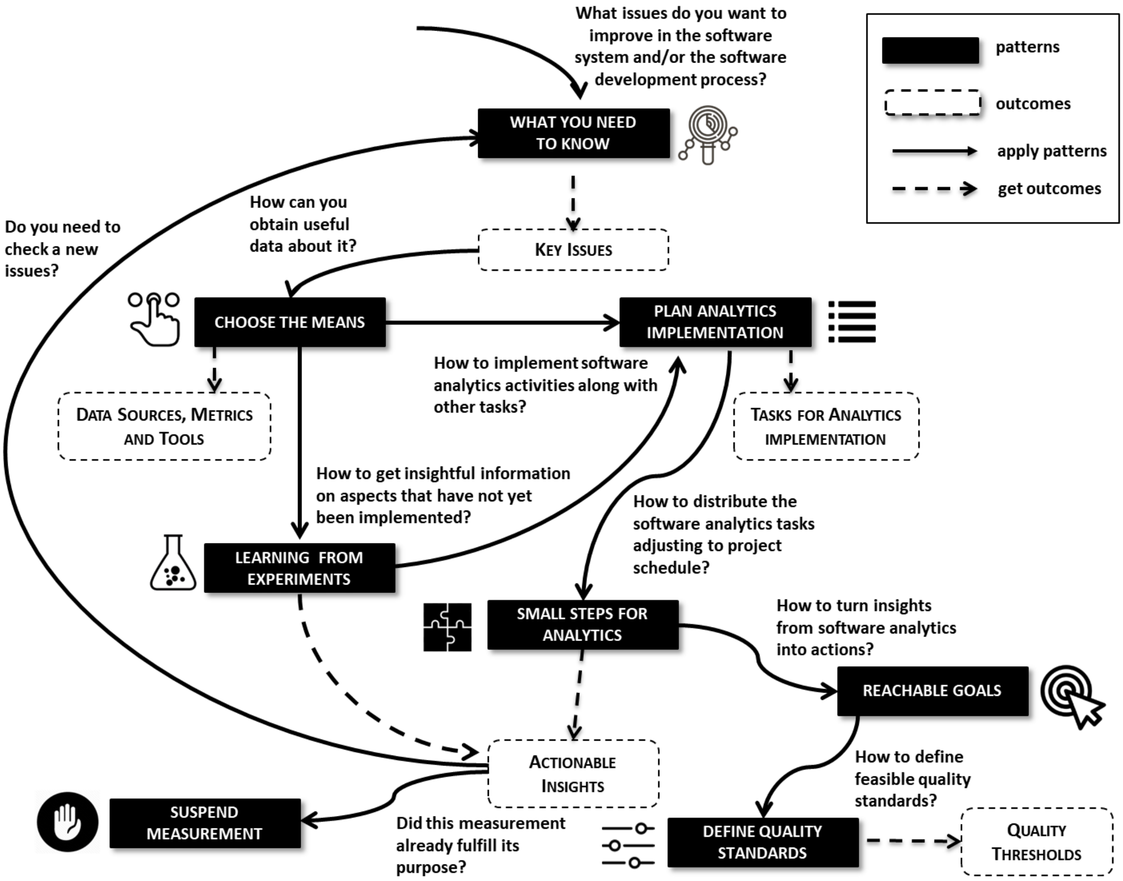
(7) Define Quality Standards: To solve how to achieve and maintain a good level of quality for important software aspects, in the context where the improvements can be made incrementally, the solution is to define quality standards and then establish minimal or maximum thresholds for any software aspect that the team intends to monitor.

(8) Suspend Measurement: To solve if an issue still need to be continually monitored after some initial measurements, in a context where the team does not yet have a monitoring system, or the current system is overloaded with other issues, the solution is to put on standby the measurements that already fulfilled their initial goal, are costly to be continuously monitored, or that do not represent a value to the team at that moment.

Figure 4.1 provides an overview of the pattern language for *software analytics* by showing how patterns relate to each other. The blocks in black represent the patterns representing steps recommended for implementation of *software analytics*. The dashed blocks represent the expected outputs from the application of the patterns. Questions included among the patterns refer to factor that motivates the application of the pattern. According to the proposed patterns, the first step towards to implement *software analytics* process is to define What You Need to Know. After that, with the purpose to answer the raised issues, the team needs to Choose the Means that will be used to data gathering and analysis. Learning from Experiments can be a way of testing a particular solution that the team is not sure if it is the best way from a practical standpoint.

During the Software Analytics Planning, the team plans the analytics activities and prioritizes the tasks in their to-do list along with other development tasks. Because analytics activities can be time-consuming, the team do not have to be deployed them at once. Then, the team can set Small Steps for Analytics, according to delivery schedule. Based on actionable insights, the team needs to define Reachable Improvement Goals to implement the improvements in software or its development process. Following a continuous improvement mindset, the team Define Quality Standards to guide their improvement actions. The team can apply the pattern Suspend Measurement when measurements no longer make sense or when they have other priorities at the moment.

Figure 4.1 - Overview of the patterns and their relationships.


What issues do you want to improve in the software system and/or the software development process?
patterns
outcomes
apply patterns
get outcomes
WHAT YOU NEED TO KNOW
Do you need to check a new issues?
How can you obtain useful data about it?
KEY ISSUES
CHOOSE THE MEANS
PLAN ANALYTICS IMPLEMENTATION
How to implement software analytics activities along with other tasks?
DATA SOURCES, METRICS AND TOOLS
TASKS FOR ANALYTICS IMPLEMENTATION
How to get insightful information on aspects that have not yet been implemented?
How to distribute the software analytics tasks adjusting to project schedule?
LEARNING FROM EXPERIMENTS
SMALL STEPS FOR ANALYTICS
How to turn insights from software analytics into actions?
REACHABLE GOALS
ACTIONABLE INSIGHTS
How to define feasible quality standards?
SUSPEND MEASUREMENT
Did this measurement already fulfill its purpose?
DEFINE QUALITY STANDARDS
QUALITY THRESHOLDS

SOURCE: Prepared by the author.

4.4 Patterns description

This section presents the patterns for *software analytics* in detail. To describe them, we used the format based on Alexander's style (ALEXANDER, 1977), which provides the pattern name, context, problem, forces, solution, examples, consequences, known uses, and related patterns.

4.4.1 What you need to know

Also known as Focus on Key Issues, Highlight Your Questions, What You Want to Improve

Development teams know that metrics and other kinds of information can be extracted from their systems in order to support decision making. Development activities generate a large amount of data. Various software artifacts including source code, bug reports, commit history, test executions, etc. could provide valuable insights about software project. There are several tools that can extract such data from the development environment at runtime. However, it is common that even development teams that have them available might not know how to use them more efficiently.

What issues do you want to improve in the software system and/or the software development process?

- Software practitioners produce different data-rich software artifacts, but they usually do not use the data to support their decisions.
- Many tools were developed to support the *software analytics* deployment, but most of them can be time-consuming to install and configure.
- For many development teams, it may be difficult to start a *software analytics* project because of lack of time, but they need to consider their benefits for software improvement.
- Some tools can generate a huge amount of data by default from the development environment at runtime, but the team should just focus on the data that will be useful to solve a given issue.

Therefore:

Define the key issues that the development team wants to focus on, in order to improve the software throughout the project.

During the development of the product, issues around software aspects arise to be solved. The team frequently make decisions to solve them based on experiences and intuitions. However, the team may not have enough information to solve a given issue. Such an issue can be related to the structure of the source code, the development process or the business rules. By defining the key issues that need to be addressed, the team can focus on efforts to solve them through a data-driven investigation in order to obtain meaningful information to support any decisions.

Some decisions might lead to one-time action, for instance, when the team needs to prioritize the implementation of an architectural component to improve the system performance. Or it might be a series of continuous decisions and actions that need to be performed through the iterations, such as for what part of the system do we need to prioritize refactoring.

As an example, imagine that a development team wants to improve their tests and need to decide where in the system they should put their effort. Using What You Need to Know, a possible question highlighted by the development team might be "What data is required to verify software test adequacy?". Answering this question, the team can avoid unnecessary data gathering, planning better on how to collect useful data to investigate the issue.

As a consequence, the team will understand the reason behind the data being collected, making better use of them. Additionally, unnecessary data will not be collected and will not take away the focus of the team on what is more important. Sometimes, tools can detect unexpected problems based on measurements that do not have a known reason. Focusing only on a subset of that information, the team can fail to notice a potential problem.

◇ ◇ ◇

Lou et al. (2013) formulated questions about incident-management as a software-analytics problem. For them, incident management of an online service requires the service provider to take actions immediately to resolve the incident, since the cost of each hour's service downtime is high. As the use of debuggers to conduct diagnosis on services is usually impracticable, the teams need to highlight other questions in order to detected anomalies and quality issues at runtime of the service.

Nord et al. (2014) presented a series of questions related to measurement and analysis for software architecture and about how to meet the business goals of software. According to them, there is an increasing need to provide ongoing insight into the quality of the system being developed. Thus, the team's questions might be, for instance, about how to improve feedback between development and deployment through means to measure intrinsic quality, value, and cost.

In the case presented by Robles et al. (2014), the information about the development effort invested in a project was considered a business strategy. And, the question highlighted by the development team was related to how to obtain software development estimations with a bounded margin of error.

◇ ◇ ◇

The pattern FIND ESSENTIAL QUALITIES (YODER; WIRFS-BROCK, 2014) focused on quality assurance is linked to this pattern. Since *software analytics* is not just addressed to solve quality issues, our pattern also encompasses other software aspects, from the development process to business requirements. As an example of supporting strategic and tactical decisions, an issue to solve could be the need for a reduction in overworks of developers. After defining WHAT YOU NEED TO KNOW, in the next step, the team need to CHOOSE THE MEANS towards to solve the related issue.

4.4.2 Choose the means

Also known as Approach to Answer, Choose the most Appropriate Means, Choose the Right Means

In order to solve the issues around WHAT YOU NEED TO KNOW, the team can conduct a data-driven investigation by collecting data related to such issues to support their decisions. If software practitioners trust only in their experiences and intuitions, they risk having a bad experience in the future because chose the wrong path.There is a plethora of data that can be collected from the development environment at runtime, which could provide concrete evidence and reasons to inform a decision making.

How can you extract useful data regarding the issues that you need to solve?

- The team can solve some problems based on their experience and intuition, but not always that decisions will be based on true premises.

- Some needs for change and improvement in software can be difficult to justify to stakeholders, but an analysis of these issues based on actual data can strengthen the team's arguments.

- Different data mining tools and methods can be used to discovering patterns in large data sets, but the team needs to know which of them are most appropriate for each case.

- There are several tools that can extract data from different types of software artifacts, but such tools need to be properly configured to extract useful data.

Therefore:

Define the most appropriate means, such as metrics, tools, techniques and other approaches for extracting data from software artifacts that will be useful in future decisions.

Focusing on the issue that should be solved, the developers can identify data that are useful in providing concrete evidence to support their decisions. This data can come from different sources – e.g., development tools, software repository, issue-tracking system, etc. As an example, developers could retrieve information about execution time from the software components in order to verify points that should be modified aimed at improving the performance of the system. As another example, commit history for bug correction, object-oriented metrics, and frequency of modification could be used to explore which parts of source code that need to be prioritized for refactoring. Yet another example could be an analysis of usage data logs for estimating the impact of a new feature.

After identifying what data is needed to explore on a given issue, the team needs to find tools and/or approaches that can be used to extract them from the system. There may be ready-to-use tools, but sometimes the team will have to develop their own tool to retrieve data in a more specific scenario. Note that, at a first moment, the team will not yet have to implement any method or setting any tool, but only identify and define the more appropriate instruments for both data collection and analysis.

A few steps can be needed to extract accurate information from the raw data initially collected. Sometimes, raw data need to be filtered, interpreted, or yet combined with other information to meet WHAT YOU NEED TO KNOW. However, the approach for this does not need to be totally defined at this point, but it is important to discuss what kind of information do you expect to have at the end.

Considering the running example, the development team wants to improve their tests and need to decide where they should put their effort. By using CHOOSE THE MEANS, developers defined that they need to extract two kinds of information: the testing coverage values and the number of commits that modified each class. From this information, they intend to prioritize classes that are highly modified and that have low test coverage. To collect testing coverage data, they can use a test coverage tool that is available in their continuous integration environment. However, they do not know a ready-to-use tool to count the commits for each class. A viable solution could be to create a script to collect and record such data in a CSV file for further analysis. In the next step, the team will need to analyzed classes with low test coverage and with a high modification frequency.

As a consequence, the team can have an overview of how they can obtain concrete evidence for a decision. From this overview, they could consider whether it worth to follow or not a *software analytics* approach by weighing the cost of the decision and the penalty for choosing the wrong alternative. Stakeholders would be able to understand better technical tasks and their impact from data analytics results. However, this process might consume precious time from the team, taking away the focus from main software development tasks.

◇ ◇ ◇

According to Pachidi et al. (2014), the collecting of usage data logs is an important means to monitor which applications are being most often used, which features were underutilized, and which features could be improved. Usage data can provide valuable information about how end-users are using the software, and whether the services are meeting their needs.

Cerulo et al. (2015) proposed the extracting data from developers' communication – as contained in emails, issue trackers, and forums – to improve the software development process.

Suonsyrjä et al. (2016) proposed a framework to support practitioners in finding a suitable technological approach for automated collecting of usage data within the process of data-driven software development.

◇ ◇ ◇

The pattern Measurable System Qualities (YODER; WIRFS-BROCK, 2014) is related to this pattern especially when the issues are related to software quality. The most common quality attributes are performance, reliability, and usability. However, any internal or external attributes related to process, business and/or resources (e.g., effort, number of coding faults found, cost-effectiveness, communication level, system structure, etc.) can be objects of measurement to solve an open issue addressed with *software analytics*. Moreover, the measurements can be used for both software evaluation and prediction. Note that, some attributes may be relatively easy to measure, while others may be difficult or costly to measure.

After Choose the Means to investigate What You Need to Know, in the next step, the team should Plan Analytics Implementation based on issues to solve.

4.4.3 Plan analytics implementation

Also known as Analytics in the Backlog, Software Analytics Planning

Towards understanding factors or causes contributing to the unwanted situation related to the software project, the team has already discussed which data is required for answering their questions and which the means will be used in the process of collecting and analyzing. The next step is implementing a plan. However, the team encounters some resistance because a task that is not related to the implementation of the software functionality may consume precious time and consequently delay the project. Particularly in agile teams, the effort in each iteration is prioritized by stakeholders. For practitioners, for instance, it may be difficult to explain to the stakeholders that the number of unresolved software issues can grow over time, generating technical debts increasingly difficult to handle and correct (LI et al., 2015). *Software analytics* implementation is a way of solving problems related to technical debts.

How to implement the *software analytics* activities fitting them to project roadmap along with other development tasks?

- *Software analytics* tasks such as the implementation of methods and configuration of tools for data collection and analysis can be time-consuming, but the team does not have to do everything all at once.
- Tasks related to the extracting and analyzing data can be plan along with development tasks, but it cannot make the team's attention off their delivery schedule.
- Data analytics can provide valuable information to help solve issues related to technical debts, but improvement actions will also need to be scheduled.
- *Software analytics* results surely lead to better decisions, but the cost of implementing it cannot be greater than the added value.

Therefore:

Add tasks related to the *software analytics* in the backlog to be prioritized with the regular project tasks.

The tasks identified when the team used the pattern Choose the Means should be planned. Tasks related to the extraction, filtering, and analysis of data should be estimated and prioritized. At this point, the team needs to consider whether the cost of implementing *software analytics* will not be greater than the value added to the product after the possible improvements.

Many analytics tasks can be selectively performed throughout the project, or when the decision making is required. For instance, considering a decision that will be necessary to be made in two months, the implementation of analytics to support that decision is not a priority for the next iteration, then it can be postponed.

In the running example, developers want accurate information in order to improve the testing process. They chose means to investigate the testing coverage and the number of commits that modified each class. As part of the planning, they estimated the time for installing the tool for measuring test coverage and found that it would not consume much time. However, they would take considerable time to create the script for extracting the most modified classes, as well as to make a crossover analysis of the coverage data with the modified classes.

Since the number of classes was not yet large at the moment, the team along with stakeholders decided to add only the task of setting the test coverage tool to the next iteration. Hence, the script creation task and data crossover analysis remained in the project backlog because were not considered a priority at that point.

As a consequence, the activities of *software analytics* will be planned and prioritized in the to-do list along with development tasks. Since tasks will be distributed throughout the project, it may be easier to manage the project without risk of delays. However, some analytics tasks defined with low priority are at risk of remain in the backlog indefinitely and never to be done. To avoid this, the stakeholders need to be aware of how that effort can bring value to the project.

◇ ◇ ◇

Defect prediction is one common application of software analytics. Taipale et al. (2013) reported the challenges of deploying a defect prediction model into practice. They proposed a defect prediction model based on different modes of information representative of the data, such as a commit hotness ranking, an error probability mapping, and an approach to the visualization of interactions among teams.

Gonzalez-Torres et al. (2011) focused on software maintenance issues that required the comprehension of project details. Thus, they proposed a visual software analytics tool for the exploration and comparison of project structural, interface implementation, class hierarchy data, and the correlation of structural data with metrics, as well as socio-technical relationships.

Minelli e Lanza (2013a) developed a visual web-based software analytics platform for mobile applications. This tool supports the mining task and uses a set of visualization techniques to facilitate the data analysis task.

◇ ◇ ◇

This pattern can be complemented with the SYSTEM QUALITY DASHBOARDS (YODER; WIRFS-BROCK, 2014) pattern since it refers to a solution that facilitates the monitoring of the software measurements. By adopting dashboards, the team can visually identify potential risks at runtime and plan action to solve or mitigate them.

4.4.4 Small steps for analytics

Also known as Analytics Tasks in Multi-steps, Analytics in Small Steps

While recognizing the value of *software analytics*, the team' priority is always to develop the target software. Some tasks to implement a *software analytics* project can be extremely complex. Software measurement implementation, the configuration of the tools in the development environment, and data analysis can be time-consuming. The set of software data can provide valuable information, but it can be hard to analyze and interpret. Receiving much information at once, the team may not be able to turn them into information with real practical value (actionable) in a timely manner.

How to implement *software analytics* in a pace that it does not to overburden the team?

- Because of the tight schedule, the team may not have time to implement *software analytics* tasks, but they are aware that technical debts tend to accumulate and make the situation more complicated.
- Some tools in default configuration may generate a lot of information about the system, but the team is not able to handle so so much information at once.
- It is important to keep an overview of the *software analytics* project and have a plan step-by-step outlined from beginning to end, but the team can go back and re-evaluate the tasks priorities whenever necessary.
- There may be big and complex analytics tasks to be done in a short period of time, but breaking such tasks into smaller tasks helps everyone on the team stay on schedule.

Therefore:

Adjust *software analytics* tasks within the team schedule by breaking down them at smaller portions to be carried out in multi-steps.

By using the pattern to Plan Analytics Implementation, the team will have an overview of the analytics tasks and planning of how to perform them step by step.

Certain tasks, however, can be complex and time-consuming. The team cannot take additional commitments on analytics that compromise their delivery schedule. However, the team would be postponing the solution of a relevant issue or failing to settle a technical debt when postponing any *software analytics* task. Instead, the team defines Small Steps for Analytics breaking down complex tasks at smaller portions to be carried out in multi-steps. This strategy can help to avoid delays and increase team satisfaction with the task done.

Some analytics tools have built-in features to support the monitoring of diverse software aspects collected from different sources. Such tools can generate a lot of information for developers at once. It will take much more time and effort to manipulate a great amount of information. In order not to overcharge the team with excessive and unnecessary information, the tool can be set to provide only specific information to complete a prioritized task.

Considering the running example, the team decided to implement the SonarQube [5] that can produce a test coverage report integrated into the continuous integration server. This tool can generate many other kinds of information, such as object-oriented metrics and bad practices detection. Based on this pattern, the team decided to disable the extraction of other kinds of information that would not be used by the team right now. Moving on to other tasks, other tool features might be enabled in the next iterations.

As a consequence, the team can move forward with the *software analytics* project without harming other development activities. Little by little, they will be able to solve key issues and decrease technical debts based on accurate information. However, critical problems can be detected late due to a slower project pace. Moreover, measurements that have been paused to avoid overload of information might be needed to support other decisions.

◇ ◇ ◇

Baysal et al. (2013b) argued that modern issue tracking systems that provide an immense amount of raw information that sometimes is irrelevant in given situation. They suggest personalized development tools that highlight only the most important information for developers by reducing information overload.

[5] https://www.sonarqube.org

Pinto et al. (2016) proposed a tool that provides architectural compliance checking as part of the continuous integration process. When violations are detected, this tool can lock the integration to the software repository.

Regarding useful information, Turhan e Kuutti (2016) state that a simpler analysis to answer a simpler question can provide more actionable insights to the team than a more complex alternative.

◇ ◇ ◇

The Plan Analytics Implementation pattern is related to this pattern because having an overview of the project is necessary to manage tasks and adjust them to fit the team schedule. Following this pattern, the team can review and prioritize tasks as needed. Also, the amount of information can be controlled.

4.4.5 Reachable goals

Also known as Actionable Insights, Achievable Targets

Metrics indicate that always something could be better. Through an automated analytics tool, the team can detect several points of improvement. However, faced with many issues to solve, the team is a risk of losing focus on what they wish to achieve. Moreover, they can have the sensation that is not moving forward when they failed to reach the minimum quality boundaries set far above reality.

How to turn *software analytics* findings into actionable insights to improve software aspects?

- *Software analytics* can generate a lot of interesting insights, but the team needs to set clear and actionable goals they want to achieve.

- The team can identify many points of improvement from analysis of software data, but they may lose focus on what are the project priorities when setting many goals.

- The result from analytics can identify a huge number of existing issues to fix, but not all issues can be solved right away, in view of the tight team's schedule.

- An ambitious goal can be achieved, but sometimes it takes it can take a long time, and the team may have the feeling that they are not evolving with regard to the improvement issues.

Therefore:

Take actionable insights from the *software analytics* findings, and from them, settle reachable goal adjusting the action steps.

Based on ideas emerged from *software analytics* with real practical value, the team should define the goals they want to achieve. For bolder goals, the team must adjust the action steps needed to achieve it. This prevents the team from becoming unmotivated every time they can't reach a goal. Both team and stakeholders should realize the benefits of *software analytics* taking into an account implemented improvements actions.

It is worthwhile noting that an improvement action sometimes does not have to be fully implemented in a single iteration. That is, the improvement actions can be planned to be carried out in multi-steps. For each step, the team sets achievable goals adjusted to their delivery schedule.

Considering the running example, imagine that through *software analytics* the team found three controllers modified frequently that had less than 20% code coverage. Now, the team can prioritize which classes need test coverage more urgently. However, the tests might be gradually implemented when, for example, this activity involves many features. As a reachable goal, the team defined to increase their code coverage to at least 50% in the next iteration. In order to evaluate the effects of their actions and manage the work in progress, the team decided to keep this issue in continuous monitoring.

As a consequence, the development team ends up establishing a culture of continuous improvement. By balancing the amount of work in progress, the team avoids accumulating uncompleted works. Sometimes, the team may not be able to deal with an extra amount of work needed to act over *software analytics* insights and take advantage of its benefits. And, if this work is not well planned, the technical debt may even increase rather than reduce.

◇ ◇ ◇

Haron e Syed-Mohamad (2015) proposed a plug-in for IDE that integrates test coverage, number of defects, number of unresolved defects, defects severity and lines of codes, aiming to provide an analytical view for practitioners to assess and validate the testing results.

As to continuously monitoring and measuring activities, Souza et al. (2015) noted that improving automated testing tools and using integration repositories are two measures that can improve any project. However, they pointed out the importance of easier access to up-to-date information about the process, in order to evaluate the impact of yet-to-be-made decisions.

As a practice of continuous inspection, Guerra e Aniche (2015) have recommended the use of static and dynamic analysis tools that retrieve information about important quality attributes from the source code, such as test coverage, complexity, and decoupling. Based on this information, the developers continuously can evaluate and refactoring small portions of code – one piece at a time.

◇ ◇ ◇

A related pattern is CHUNKING (WEISS; MOCKUS, 2013) that shows how to conduct an analysis of the set of changes made to a software system over time. This analysis aimed at identifying sets of code that have the property that a change that touches a chunk touches only that chunk. This pattern may be useful to help the team coordinate and optimize its improvement actions. In this pattern, authors provided an algorithm to identify uncoupled pieces of software (chunks) where each one represents a module on which an individual or a small team can work independently.

4.4.6 Learning from experiments

Also known as Run Experiments, Measure with Experiments

The issues that emerge in *software analytics* can be related to different needs. At the development process level, the team may need to evaluate for instance new methods, tools, or practices. At the product level, the team may need to evaluate the requirements, features, or usage data. At the user experience level, they may need to evaluate product usability, user satisfaction, design aspects, etc.

Some of these issues can be investigated through data collected from the development environment and software artifacts such as source code, bug reports, test cases, usage logs, documentation, etc. For other issues, however, the team may need to evaluate the usability of a feature that has not yet been implemented. Even further, the team may have implemented a feature that needs to be improved but does not know how to improve it. In both these situations, the team has nowhere yet to collect and analyze data to support their decisions.

How can we obtain information to make informed decisions about software issues on some aspect we have not yet implemented or we need to redesign?

- The team often has more than one way of implementing the same feature, using different methods and tools for development, and adopting different alternatives of architectural design, but they need to seek to make their choices as successful as possible.
- Sometimes the team makes a decision that will be simple to reverse if it does not work out, but some solutions are worth experimenting before implementation to avoid wasting resources and rework.
- Experiments can fail, but the team can improve the experiment design by analyzing what went wrong.
- Sometimes, experiments can produce inconsistent results, but the team can investigate the cause of such inconsistencies and then conduct new experiments if it proves feasible.

Therefore:

Create an alternative solution and perform an experiment collecting data that allow the comparison with the current solution.

Experiments allow us to test our hypotheses for better decision making. Results from experiments can provide us with relevant information to find the best alternatives for design, tools, approaches to development, test methods, among others. Through experiments, we can compare two different approaches, where the control group can be an existing solution in use. That is, we can verify whether an idea is promising or it makes no sense to continue with it.

Moreover, experiments can have a low cost of implementation. However, before opting for experimentation, the team always needs to weight the cost of doing one or more experiments with the cost of re-engineering or redesigning after implementing something. The team needs to have a clear purpose for the experiment and have a reasonable hypothesis to test. The experimental design should be carefully planned, and the experimenters should know which aspects of the software or process will be observed.

The results should be analyzed without bias by the development team to ensure successful experimentation. During the experimental design planning, the team should be especially careful also to define the experiment size. Large experiments can be costly and unfeasible. Both ROI and the time to implement an experiment are important factors to be considered by the team before adopting them.

Sometimes experiments may not work out, and sometimes they can produce conflicting or unclear results. When an experiment did not provide useful information or has unclear results, the team decides if new experiments should be carried out from the lessons learned. Replicating an experiment may be impractical depending on its cost and size. Small experiments tend to be cheaper and easier to replicate. About experimenting and learning, worth taking into consideration Linda Rising's advice:

> You can't realistically plan anything from the beginning; the only way to reach your long-term goals or solve your big problems is to try a small thing and learn from the experience. That's how we have always learned. Babies do this from the start. It's the basis for the scientific approach. Experiment and learn (RISING, 2011).

As an example, imagine that a development team wants to increase the number of hits/clicks on related products in an e-commerce application. Currently, in this application, the products are merely recommended according to the category. The team has the following idea: an algorithm to recommend products that were recently bought with the product being searched. Additionally, the team wants to rearrange products on the user interface. They do not know how much it will be pleasing to the end-user. In order to save time and effort, the team develops some prototypes and runs an experiment with a limited number of users, representing the target audience. From the experiment results, they were able to know which was the best option for the user interface redesign.

As a consequence, the results of experimentation can produce insightful information about the product or the development process, that is, the reliable and valuable knowledge needed to make better decisions. Sometimes, team members can be biased in how they interpret the results of an experiment. If it doesn't produce the results they expect, they may discount the results or find ways to invalidate the experiment. Other times, the results of an experiment may be inconclusive. In that case, the team must decide whether to perform another experiment to pick among equally viable options. Also, an experiment can provide misleading information which did not test the hypothesis. In that case, the team may have to spend time figuring out why something you thought would improve the system did not.

◇ ◇ ◇

Kim et al. (2016) investigated the competencies and working styles of data scientists in software-oriented data analytics context. They reported an increase in the randomized two-variant experiments called A/B testing in order to assess the requirements and utility of new software features. Because of numerous possibilities for alternative software designs, data scientists along with engineering teams have built software systems to inject the changes, called flighting.

Gousios et al. (2016) introduced the concept of streaming software analytics and proposed a data analytics infrastructure which unifies the representation of historical and current data as streams and enables high-level aggregation and summarization using a common query. This approach can facilitate the execution of experiments, as well as data collection and correlation of the results of applying specific design and development decisions and their outcomes.

Liechti et al. (2017b) introduced the idea of test analytics with the purpose of helping an agile development team to improve their test process. Focusing on collaborative practices, they organized a series of workshops to train the team and started with small experiments on simple features which allowed to evaluate and select a set of tools and to create a collection of examples.

◇ ◇ ◇

Build Prototypes (COPLIEN; HARRISON, 2005) is a related pattern that highlights the usefulness of prototypes in experiments to understand requirements, validate requirements with customers, explore human/computer interactions for the system, or explore the cost and benefits of design decisions.

Early Validations is another pattern addressed for software startups (MELEGATI; GOLDMAN, 2015). In that pattern, experiments are performed to validate or reject an initial hypothesis and make decisions about the direction of the project based on acquired knowledge.

4.4.7 Define quality standards

Also known as Define Quality Boundaries, Set Quality Thresholds

By using an analytics approach, the team collects data about a given issue. From the analysis of data, they discuss possible solutions and have insights to take action on. Based on these insights, the team defines which goals they want to achieve concerning critical issues. The improvement actions more audacious can be implemented in a stepwise fashion. However, the team needs to establish milestones to achieve the goals. Once implementing the improvements, the team can evaluate the impact of the changes by collecting feedback from stakeholders. Once the goals have been achieved, the challenge will be to maintain the quality level achieved.

How you can achieve and maintain a good level of quality for important software aspects?

- By analyzing software data, developers can make better decisions about improving the development process and software quality, but the quality of some aspects will need to be continuously monitored.
- The culture of continuous improvement is stimulated by achieved goals and satisfaction of stakeholders, but it may not be easy to convince stakeholders about the tradeoffs of continuous inspection.
- The process of continuous improvement helps sustain the software evolution and maintenance; the team must have reachable goals and define quality standards.

Therefore:

Define quality standards and then establish minimal or maximum thresholds for any software aspect that the team intends to monitor.

When investigating a given issue, the team might obtain information needed to take steps to solve it based on actual data.

To maintain the level of quality achieved, some software aspects can be monitored longer. For this purpose, the quality metrics can be used to assess different software aspects such as code quality, testing coverage, performance, bug fixing, productivity, and user satisfaction.

Following a quality standard, the team should establish quality thresholds to aspects that they want to monitor. Quality thresholds can be defined whenever there is a need for continuous inspection. For example, for issues related to coverage testing, the response time cannot exceed 2 seconds (maximum acceptable value) or the test coverage must be at least 70% (minimum acceptable value). However, the minimum or maximum value established for an attribute should be periodically analyzed and can be redefined focusing on continuous improvement. Moreover, the team may need to make trade-offs between different software aspects – e.g., performance, security, and usability. Thus, the threshold value of an aspect can be redefined so that another aspect can work.

As an example, let us suppose that the team wants to automate more of their tests, but they do not know where to start, once there is an immense amount of classes. Their key issue is "Where should we focus our test efforts?". To answer this question, they identified the need to investigate two data sources: the code-source to verify current test coverage, and the code repository to verify the percentage of commits related to fixing bugs and the classes with the highest number of changes to identify the classes with more problems. As data-gathering mechanisms they decided (a) to adopt SonarQube for code coverage; (b) to find a tool to collect the number of changes, and (c) to develop a script to relate commit messages with bug issues. When analyzing the collected data looking for insights, the team found that "Web controllers have a high change rate and a low coverage" and "many changes in DAOs were related to bug fixes". Then, as part of an incremental goal, they established a minimum class coverage for Web Controllers of 60%; and minimum class coverage for DAOs of 80%. Moving forward, they set a threshold of at least 80% coverage for new classes.

As a consequence, as new requirements come in, the team is engaged in evolving the software, while maintaining a quality standard. By setting the boundary values, the team assumes a commitment to meet some pre-established quality standards. If these boundaries do not comply, the team must identify the causes because they have failed and if necessary redefine new achievable goals. Sometimes, the team attempting to reach a certain target can unknowingly fail to notice other issues.

◇ ◇ ◇

Feldt et al. (2013) reported how visualization and correlation between test and code measurements can support decisions on software quality improvements, based on a case study where heatmaps were employed to visualize and monitor changes and identify recurring patterns of an embedded control system.

Foidl e Felderer (2016) discussed the challenges related to defect prevention and highlighted the importance of software analytics to improve quality assurance of Internet of Things (IoT) applications. They recommend the usage of data mining algorithms and techniques (e.g., classification, association, and clustering) as well as predictive modeling to support the quality assurance of IoT solutions.

Martínez-Fernández et al. (2018) proposed a quality model called Q-Rapids. The model encompasses four quality aspects. Maintainability, reliability, functional suitability are quality aspects from ISO 25010 and refer to the quality of the software system, while productivity refers to the quality of the software development process. In this model, quality aspects are calculated based on product and process factors. Both product and process factors are calculated based on the assessed metrics. And, metrics are calculated from raw data, which may come from heterogeneous data sources.

◇ ◇ ◇

The System Quality Dashboards pattern (YODER; WIRFS-BROCK, 2014) recommends the use of dashboards to monitor important qualities aspects from values established by the team. Tools for monitoring systems such as SonarCube allow you to configure alerts and notifications when measured values cross a threshold. The Continuous Inspection pattern (MERSON et al., 2014) captures the overall practice of continuous inspection to preserve the quality of the source code and its alignment to the architecture in an agile environment.

4.4.8 Suspend measurement

Also known as Standby Measurement, Hold Measurement

When investigating a critical issue using *software analytics*, the team first performs data gathering to obtain accurate and in-depth information about it. And then, they analyze the data in order to find out the best way to solve that issue.

In some instances, the information may be insufficient and the team will need collecting more data. Sometimes, a greater effort may be required to automate data collection and analysis when the team defines that a particular issue needs to be monitored continuously. However, there may be issues with a low likelihood of recurrence; or even issues that are no longer a priority for the team.

What should the team do when a particular problem requires additional effort to be continuously monitored, or when a problem has a low likelihood of recurrence, or whenever a particular measurement is no longer a priority at that time?

- It is crucial that the development team make decisions in its process based on evidence, but the *software analytics* activities need to be carefully planned and evaluated over time because they can consume a lot of team effort.

- Critical issues need an immediate investigation to avoid software operation failures and information inconsistency, but the team has other priorities in the project.

- A given issue that has been the subject of measurement may have achieved the primary goal and, for the next interactions new measurements will be unnecessary. Achieve a goal could be to prioritize the software maintenance tasks by identifying classes with the highest number of bugs, for example.

- Once a metric has been obtained through scripts, such as a static log analysis or a SQL query, it can be costly to add it into a continuous monitoring mechanism or to execute it frequently. The integration of this measurement in the deployment or build environment collecting live data might demand a considerable effort.

- It may be difficult for the team to maintain continuous monitoring of a particular aspect of the software. However, the amount of value-added may outweigh team effort. Moreover, some tools can facilitate the process of automating that reducing the effort significantly in the continuous monitoring implementation.

Therefore:

Put on standby the measurements that already fulfilled their initial goal, are costly to be continuously monitored, or that do not represent a value to the team at that moment.

When facing problems related to software usage, for instance, the team may need to check specific issues. The team is suspicious of some flaws in the system, but they have no idea about the dimension of the problem, nor the real impact upon system operation. They decide to investigate the issue through further analysis. In one-off action, they detect the problem through *software analytics*. Then, they define the next steps to solve the problem and put the process used to collect and analyze information on hold. The team can suspend measurement of the issues with a low possibility of recurrence. However, some issues may need monitoring for an extended period in order to prevent any flaw in the system operation. At that moment, however, the team has defined that, for some reason (e.g., effort, cost or other project constraints), the monitoring of these issues cannot be implemented immediately.

When investigating and detecting potential problems, the team can choose metrics to monitor them continuously. But, in many cases, the team may not have a monitoring system yet, or it could be that the existing system is overloaded due to monitoring other issues that are more important. Or even, the monitoring system does not provide resources to monitor a particular problem. Anyway, the team will have a certain cost to prepare the monitoring system.

The cost of collecting data either manually or in a "one-off" way can also be very costly for the team. For instance, consider data extracted from the database using an SQL query or information extracted from a log analysis using a simple script. Both solutions need to be developed and implemented by the team. However, the team needs to assess the feasibility of measurement of a given issue, taking into account the list of priorities and the cost of implementation.

As a practical example of when to Suspend Measurement, let's suppose that a team is using *software analytics* to know about the consistency of the information stored in the database where data are provided daily (minute by minute) from a distributed sensors network. The team suspects data inconsistency caused by sensor failures, but they do not know the extent of the problem.

From the analysis of data, the team has obtained evidence to prove their suspicions and make some decisions about what to do to resolve this problem. In contact with the domain experts, they discuss mechanisms to normalize the data before the information is delivered to the end-user of the application. They envisage the possibility of implementing continuous monitoring on the issue, but currently, they have other higher priorities and demands. Because of this, they decide to put the measurements on hold. As an advantage, after this experience, the team already knows how to collect and analyze sensor data any moment they need to in the future.

As a consequence, the team should act quickly to gather evidence on issues of concern, which can have an irreparable effect if they are ignored for a long time. As an immediate action, the team can mitigate a problem by using corrective methods. In the future, they may implement preventive actions. Knowing how to do it, the team can use the same means to fix a problem when it occurs. However, the team can forever postpone a definitive solution of a problem, the installation of an alert program, or the implementation of a continuous monitoring system for the most critical issues.

◇ ◇ ◇

Shull (2014) presented a discussion between two ways to handle software data. The first advocate that analysts need to be intentional and to work on what is useful, not just what is convenient to collect, and the other argue for reflecting and learning more from collected data before collecting something new.

Ram et al. (2018) reported challenges faced in operationalizing metrics based on multiple case study conducted at four Agile software companies. The main challenges found were the lack of data or appropriate tools to produce that data, existing process inhibiting change, and difficulty in deriving actionable inputs.

Huijgens et al. (2017) investigated how strong metrics for agile (Scrum) DevOps teams can be set in an iterative fashion strong agile metrics. Strong agile metrics refers to metrics with high predictive power to be able to support a highly effective monitor and control capability within continuous delivery context.

◇ ◇ ◇

The Recalibrate the Landing Zone pattern (YODER; WIRFS-BROCK, 2014) is related to this pattern by addressing the implementation of decisions when resources are incrementally implemented. It is natural that some criteria adopted in the course of the project need to be adjusted over time. These decisions can affect or limit the ability to achieve new goals and meet other demands. Thus, measurements may be provisionally suspended and then refined in future actions. The Architectural Trigger pattern (WIRFS-BROCK et al., 2015) suggests that when the team does not know when to evolve the architecture, they can develop architectural triggers. Similarly, the team can define triggers to warn them when a certain condition may require immediate action to treat a particular issue.

4.5 Chapter summary

This chapter presented a pattern language for supporting *software analytics* activities implementation in practice. The proposed patterns focus on recurring solutions about how to incorporate *software analytics* on an interactive and continuous basis to inform the decision-making process of software practitioners. Considering different levels of decision-making, these patterns can address different issues by using a *software analytics* approach, such as the ones related to source code quality, testing methods, bug treatment, runtime software properties, reuse of components, maintenance and software evolution, development practices, teamwork and productivity, customer and requirements, user experience, and services.

5 SOFTWARE ANALYTICS CANVAS

This chapter reports the design cycle of the artifact proposed to supporting agile teams in the *software analytics* activities, named SA Canvas. This artifact was based on the recommendations covered in the pattern language described in the previous chapter. The evaluation cycle of the artifact will be reported in Chapter 6.

Section 5.1 contextualizes how the idea of consolidating patterns for *software analytics* into an artifact emerged with the aim of supporting software practitioners in a more practical way. Section 5.2 presents the investigation findings on information flows and the role of artifacts in agile environments. Section 5.3 covers the measurements in an agile context. Section 5.4 introduces the canvas as artifact to support *software analytics* activities. Finally, Section 5.5 describes the first version of the SA Canvas, some recommendations, and an example of use.

5.1 Background

The idea of the artifact to support analytics activities based on the patterns introduced in the previous section was triggered from feedback and insights obtained in meetings with members of the EMBRACE team at the Space Weather Center, in October 2017. EMBRACE was created in 2008 at the National Institute for Space Research (INPE) as a space weather forecast and information center in Brazil through the association of the Space and Atmospheric Science division (CEA) and of the Laboratory of Computation and Applied Mathematics (LAC). The purpose of EMBRACE is to provide relevant information on spatial phenomena capable of disrupting economic activities. Such information has a great scientific and technological impact and can assist decision-making by both the government, regulatory agencies, and Brazilian companies.

Through a web portal [1], EMBRACE researchers offer different products based on their research in space climate. On the website, information on disturbance scales by Solar Radiation (X-Ray Flow or R Scale) and Geomagnetic Storm (Ksa Index or G Scale) are available to fulfill the demand of the community of scientists working with satellite orbit control, communication in the HF frequency range, and electric power operators, for example. Over the years, EMBRACE researchers and developers have invested in updates and expansions of their information systems (databases, servers, physical and virtual memories, and software), always focusing on security and quality.

[1] http://www2.inpe.br/climaespacial/portal/the-embrace-program/

Due to the domain nature, a large volume of data is collected daily by a sensor network maintained by EMBRACE. Sensor data usually are used in the models developed by the researchers and feed the databases at the research center. The reliability and consistency of the information available on the EMBRACE portal are aspects very relevant for them.

When we presented the *software analytics* patterns to the EMBRACE team, they showed great interest in applying them in their practice. The team saw the *software analytics* as an opportunity to guide their decisions during the process of developing new products, as well as the evolution and maintenance of legacy applications. On that occasion, we interviewed the Product Owner (PO) to understand their daily work and agile practices adopted by team. We found that the developer team was small composed of one back-end developer and two front-end developers. The members were hired from a third-party company. The PO presented us a new product under development and comment on difficulties to maintain the legacy products. While the new products were already being developed within good development practices and with automated testing, the legacies had no test coverage. As for software metrics collection, he commented on the difficulty to set up rules on SonarQube [2], a tool for inspecting the code quality. SonarQube provides a pre-configured dashboard that present output from various sensors but offer very limited customization capability (DEISSENBOECK et al., 2008).

After this interaction, we perceived that a *software analytics* project could encourage the team to take the time to identify the most appropriate tools and means to establish a more efficient measurement program. As for legacy software, *software analytics* could help them prioritize and optimize their maintenance activities. From that meetings, we started to devise how the patterns could be applied in practice in a systematic way. With this objective in mind, we decided to investigate the existing literature to understand on how information flows and what is the role of artifacts in agile environments (Section 5.2). In addition, we also investigated the studies on measurements in agile contexts (Section 5.3).

5.2 Information flow and the role of artifacts in agile environments

Agile principles advocate that the most efficient and effective method of conveying information to and within a development team working in close collaboration is face-to-face communication (BECK et al., 2001).

[2] https://www.sonarqube.org

Two of the four values listed in the agile manifesto highlight collaboration as a key practice for agile teams. Within Agile environments, collaborations are "complex events that happen through talk, through artifact use, through gestures, through various electronic media, or through combinations of these channels of communication" (BROWN et al., 2011).

Regarding channels of communication, one of the primary practices of XP, for instance, is keeping an "informative workspace" about teams' work where anyone interested can be able to get a general idea of how the project is going, expending very little energy to view what is displayed (BECK; ANDRES, 2004). The information displayed in agile workplace tend to change over time following the dynamism of the team's activities (OLIVEIRA et al., 2013). Artifacts like posters, boards, or displays with colored sticky notes posted in a place where people can see it as they work are considered an important *information radiator* (COCKBURN, 2004) which allows more communication with fewer interruptions since everyone can access any time the shared information.

Seemingly simple, the agile artifacts are used in disciplined and sophisticated ways allowing an efficient means of communication and work organization. By analyzing in detail the activity of one agile team, Sharp et al. (SHARP et al., 2006) found that the role of physical artifacts is largely restricted to process issues, rather than detailed information about the software under production. For example, story cards – a small index card used to write users' requirements – when displayed upon whiteboards, flip charts, or a wall allow the team to view the project status concerning work in progress, in testing, or done.

Agile teams use a spectrum of communication ranging from informal communication (face-to-face) to formal communication, one that is mediated by formal artifacts. Gerard et al. (GERARD et al., 2018) introduced the concept of fuzzy artifacts to include the artifacts which are not formally documented, but which are explicitly recognized by an agile team. User story [3] planning, user story specification, and Go/No-Go decisions are examples of fuzzy artifacts identified in the study by Gerard et al. User story planning refers to the assignment of priorities and effort using, for instance, the planning poker technique (COHN, 2005), while user story specification refers to any additional information requested by the designers. Go/No-Go decisions are decisions about, whether or not, the ongoing project should be continued.

[3] User stories are requirements notation commonly adopted in agile development (COHN, 2004).

5.3 Measurement in agile context

Agile was designed to reduce the cost of change and uncertainty (KS, 2017). However, due to the urgency of product delivery, lack of time, and other constraints, many agile teams have neglected the systematic use of metrics to improve the product quality or the way they develop software (RAM et al., 2019). Within this scenario, practitioners end up making important decisions based on vague assumptions or past experiences. Nevertheless, to save time and resources, decisions must be as accurate as possible.

Currently, there are numerous tools to support *software analytics* when it comes to collecting, consolidating, and analyzing data. The range of tools includes unit testing frameworks, software metrics tools, bug checkers, dashboards for supporting developers' daily activities, and some *software analytics* platforms to be ready to go (BAYSAL et al., 2013a). Deissenboeck et al. (DEISSENBOECK et al., 2008) identified four categories of tools according to usage dimensions: (i) sensors; (ii) system analysis workbenches; (iii) project intelligence platforms; and (iv) dashboard toolkits.

Sensors include verification and testing tools, anomaly detectors and metric calculators that perform fully automated analyses. System analysis workbenches usually are used on-demand during system inspection or review to support experts in the analysis of various development artifacts such as source code and architecture specifications. Project intelligence platforms are designed to operate autonomously collecting metrics within a software development environment, offering source control, issue tracking functionality, and sensors that gather data and transmit it to a central server for analysis, aggregation, and visualization. Dashboard toolkits provide libraries to build custom-made analysis dashboards for quality analysis and project controlling. The analysis results can be visualized in a variety of formats, including general-purpose lists, tables, graphs, charts, or treemaps. SonarQube[4] is an example of a pre-configured dashboard that present output from various sensors but offer very limited customization capability (DEISSENBOECK et al., 2008).

Although many of these tools provide structure on top of which data-driven improvement processes can be implemented, some tools can be difficult to use and time-consuming to set up them (LIECHTI et al., 2017a). Furthermore, software practitioners can even get lost with so much information that existing tools can provide.

[4] https://www.sonarqube.org

In some instances, there may be a need to develop a tool from scratch in order to investigate more specific issues within certain domains (BAYSAL, 2013). However, whatever the case, the cost of extracting important information and insights from data sets using analytical reasoning, and then deliberation of improvements must be computed. As mentioned previously, to successfully run *software analytics* projects, the entire team needs to be engaged and aware of the analytics process; the software development tasks cannot be stopped to implement analytics; any measurement should be done for a known reason; and awareness that decisions informed from analytics should generate actionable goals. Based on these assumptions, an approach to supporting in an efficient manner agile teams is necessary throughout the planning and execution of *software analytics* activities, as well as for deliberating improvements.

5.4 Canvas model

Taking into account the existing studies on how information flows in agile environments, the types of artifacts commonly used to support interaction among team members, and the agile measurements practices, we choose to create an artifact in a canvas format to consolidate the concepts of our pattern language for planning and tracking *software analytics* activities.

Canvas models are visual maps structured and preformatted commonly used to improve analyzability and communicability by supporting the teamwork within collaborative environments (COES, 2014). Canvas as a powerful visual tool has been popularised by Osterwalder and Pigneur (OSTERWALDER; PIGNEUR, 2010) who proposed a canvas called Business Model Canvas (BMC) to mold a business model framework. BMC has business-level applicability and, as defined by the authors, the business model goal is to describe "the rationale of how an organization creates, delivers, and captures value" (OSTERWALDER et al., 2005). BMC has been successfully applied and adapted for the development of new business, conception and planning of projects, strategic alignment of projects, value proposition, acceleration of startups, among others (MAURYA, 2012) (JOYCE; PAQUIN, 2016) (NAGLE; SAMMON, 2016) (NIDAGUNDI; NOVICKIS, 2017).

From a practical point of view, agile artifacts tend to be simple and their main purpose is to support communication and teamwork. In this sense, a canvas to supporting agile teams in *software analytics* activities is aligned with this purpose because it should be working as an important channel to communicate the analytics process ongoing, results, insights, and decisions of the team.

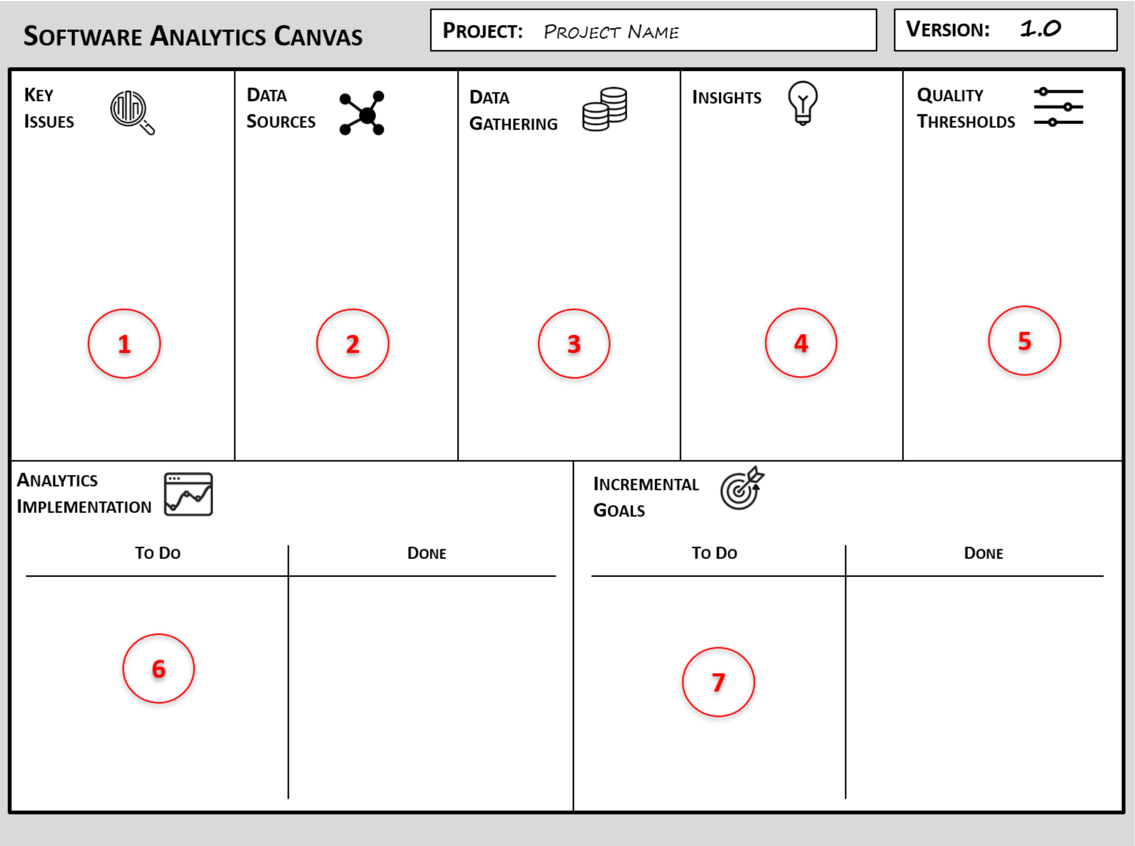
5.5 Software analytics canvas - 1st version

This section presents the first version of the canvas named Software Analytics Canvas (SA Canvas), which was initially published in (CHOMA et al., 2019). As aforementioned, the goal of the canvas is to support agile teams during planning, driving, and tracking of the *software analytics* activities for informed decision making. Next, we present the first template of the canvas. Then, the elements that make up the canvas are described. And finally, an example of the canvas usage is presented.

5.5.1 Software analytics canvas template

Figure 5.1 shows the first version of SA Canvas – a template built with seven blocks on which inputs and outputs of the *software analytics* process can be planned and managed by the team.

Figure 5.1 - SA Canvas [version 1.0].



SOURCE: Prepared by the author.

The first template of the SA Canvas was drawn on the digital format to be reproduced an A4 size picture. Nonetheless, note that the proposed canvas can be presented in another digital format (e.g., power-point, iPad app, wiki), and printed on another type of physical artifact (e.g., A3 paper, Flip-Chart, whiteboard). Or even, it can be adapted to a collaboration tool that organizes projects into boards such as Trello[5], Mural[6], and Miro.com[7].

5.5.2 Software analytics canvas elements

During artifact design, we have taken into account that it should be flexible enough to communicate the practitioners' ideas, from a broad view of the elements that compose it, including knowledge of the relationship between these elements. The patterns presented in Chapter 4 were the main basis for defining the seven canvas' elements, which also are referred to as blocks or sessions. The seven elements of the SA Canvas in its first version were named as follows: Key Issues, Data Sources, Data Gathering, Insights, Quality Thresholds, Analytics Implementation, and Incremental Goals. Next, the explanation of each canvas' element is presented including the patterns related. Also, a guiding question is provided for each item, with the objective of helping beginner users.

Key Issues. This element is related to pattern What You Want to Know. As a first step, the team can raise issues that need to be verified, analyzed and improved. The issues can be for example related to the internal quality of the system (e.g., code quality), external quality of the system (e.g., performance, bug density, the effort required to fix defects), productivity (e.g., effort estimation), and/or usage patterns (e.g., usability, user satisfaction). The guiding question is: *What does the team want to know?*

Data Sources. This element is related to patterns Choose the Means and Learning from Experiments. After defining the *key issues*, the team identifies what kind of data is needed to know more about the issue raised, and from which sources the data should be extracted. For example a dump of the database system on recent transactions, source code, behaviors' user, historical data about bugs incidence, etc. For certain issues, the team may recognize the need to collect data from multiple sources for cross-referencing. The guiding question is: *What data sources can provide information on the issues raised?*

[5] https://trello.com/

[6] https://www.mural.co/

[7] https://miro.com/

Data Gathering. This element also is related to pattern CHOOSE THE MEANS. After identifying the data sources, the team should decide which metrics and tools will be used to gather the data. The team can, for example, enable the collection of specific code metrics in the development environment, export from SGBD data referring to a given period, verify the need to create a specific script to extract data from the software repository, etc. Furthermore, the team needs to decide which methods and tools will be used to analyze the data collected. The team can employ simple statistical methods (e.g., descriptive and inferential statistics) to more sophisticated methods (e.g., data mining, natural language processing, machine learning, etc.), according to the type and amount of data. Also, the team needs to decide on the tools to support their analysis. They can opt for platforms to *software analytics* (e.g., Kiuwan[8], Seerene[9]) which can be configured according to the team's needs. The guiding question is: *How will the data be collected and analyzed?*

Insights. This element is a trigger for the pattern REACHABLE IMPROVEMENT GOALS. The team analyzes the results obtained from the collected data and discusses possible solutions and insights to making-decision. For example, the team finds that "tests have low coverage in module X", "customers prefer this approach" and so on. Then, Insights are raised from the search for solutions. Notice that, sometimes, the data analysis did not reveal significant information about the issue raised. So, the team will decide whether to continue the investigation by collecting new data or if the issue is disregarded, once no action is necessary. The guiding questions are: *What insights emerged after analysis of the results?*

Quality Thresholds. This element is related to pattern DEFINE QUALITY STANDARDS. When implementing the improvements via informed decision-making, the team can evaluate the impact of the changes by collecting feedback from stakeholders. From collecting feedback, the team will have enough information to decide whether to consider the issue resolved, or whether the issue should be monitored for longer. Concerning unresolved issues, the team will decide whether they will be re-analyzed from new data, or discarded. Ideally, the team should establish the quality threshold values for any issue that the team decides to evaluate or to keep in monitoring. For example, in issues related to coverage testing, the response time cannot exceed 2s or the test coverage must be at least 80%. The question is: *What are the acceptable values for maintaining quality standards?*

[8] Source: https://www.kiuwan.com/)

[9] https://www.seerene.com/

Analytics Implementation. This element is related to both patterns Software Analytics Planning and Analytics in Small Steps. The team should plan how to conduct analysis activities to seek meaningful information from collected data. These activities should be included on the to-do list and prioritized along with the other development tasks. In order to avoid overloading the team, such activities – that also includes the preparation of the analytical infrastructure – can be distributed throughout the project and executed by steps resulting in something deliverable. This block is divided into two regions, the first for the works to be done, and the other to control the work done. The guiding questions are: *How will software analytics activities be implemented along with other tasks?*

Incremental Goals. This element is related to pattern Reachable Improvement Goals. However, if the current goals have been fulfilled, the pattern related is Suspend Measurement. From their insights, the team discusses and defines reachable goals to put their solutions into practice, considering that these improvements can be made incrementally. Therefore, the most important in this step is to define where the team wants to reach and what goals they want to achieve. The main question is: *What are the possible improvement actions to be implemented?*

5.5.3 SA Canvas: a fictitious example

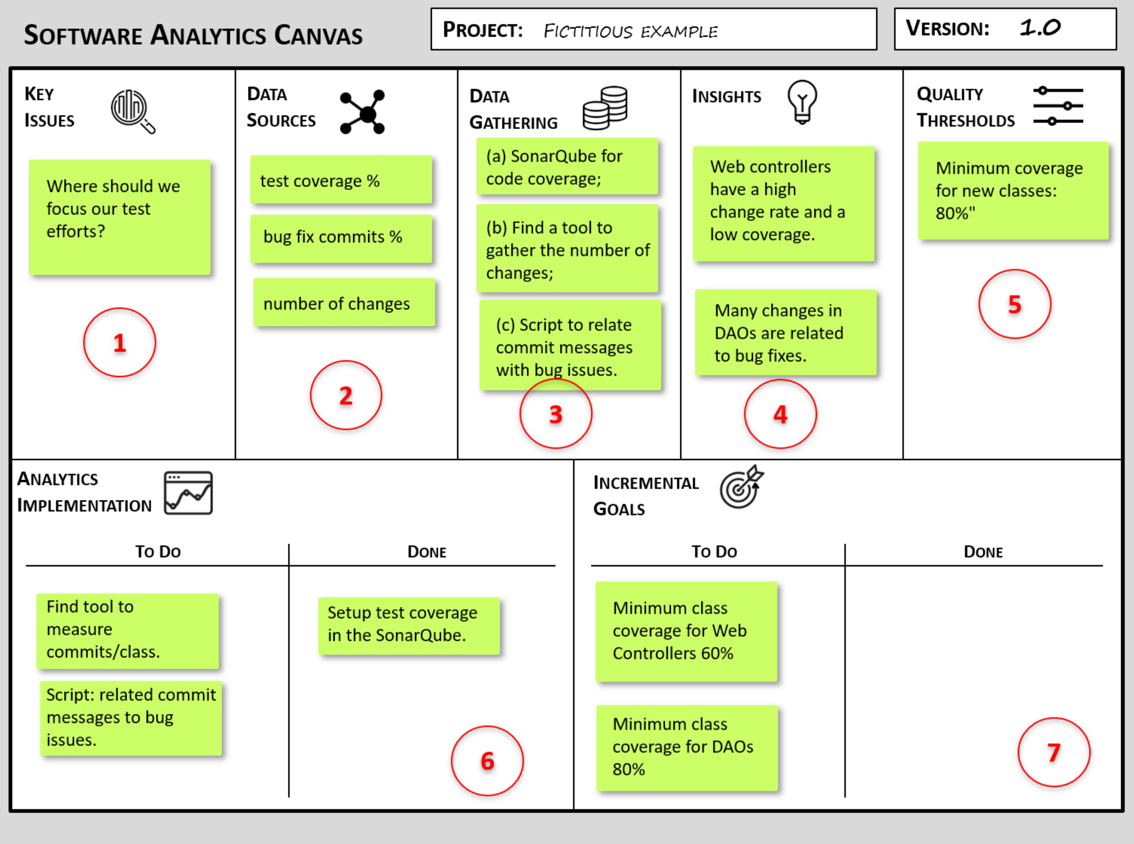
To better understand how SA Canvas works, this section presents an example of how it can be used. Figure 5.2 shows a fictitious example based on a very common issue in the area of software development. Supposing the team wants to automate their tests, but they do not know where to start, once the software has an immense amount of classes.

As showed in Figure 5.2, an example of *key issue* should be "Where should we focus our test efforts?". To answer this question, the team identified the need to investigate two *data sources*: the code-source to verify current test coverage, and the code repository to verify the percentage of commits related to fixing bugs and the classes with the highest number of changes to identify the most "problematic" classes. As *data gathering* mechanisms they defined (a) to adopt SonarQube[10] as the tool for inspecting the code quality; (b) to find a tool to collect the number of changes, and (c) to develop a script to cross-reference committing messages with bug issues.

[10] See more at https://www.sonarqube.org/

When analyzing the data looking for *insights*, the team found that “Web controllers have a high change rate and a low coverage” and “many changes in DAOs are related to bug fixes”. Then, as *incremental goals*, they established a minimum class coverage for Web Controllers of 60%; and minimum class coverage for DAOs of 80%. And, for new classes coverage, they defined as *quality thresholds* a minimum coverage of 80%. In the current status of this fictitious project, the team is *implementing software analytics* in an incremental way. Thus, they have already *done* the “setup test coverage in the SonarQube”, and now *to do* they need to “find a tool that measures commits/class” and to develop the “script for relating commit messages to bug issues”.

Figure 5.2 - SA Canvas [fictitious example].



SOFTWARE ANALYTICS CANVAS

PROJECT: FICTITIOUS EXAMPLE

VERSION: 1.0

KEY ISSUES

Where should we focus our test efforts?

DATA SOURCES

test coverage %

bug fix commits %

number of changes

DATA GATHERING

(a) SonarQube for code coverage;

(b) Find a tool to gather the number of changes;

(c) Script to relate commit messages with bug issues.

INSIGHTS

Web controllers have a high change rate and a low coverage.

Many changes in DAOs are related to bug fixes.

QUALITY THRESHOLDS

Minimum coverage for new classes: 80%"

ANALYTICS IMPLEMENTATION

TO DO	DONE
Find tool to measure commits/class. Script: related commit messages to bug issues.	Setup test coverage in the SonarQube.

INCREMENTAL GOALS

TO DO	DONE
Minimum class coverage for Web Controllers 60% Minimum class coverage for DAOs 80%	

SOURCE: Prepared by the author.

5.6 Chapter summary

This chapter introduced the types of artifacts commonly used to support interaction among agile teams and the practices of agile measurements. Considering the artifacts and practices that characterize agile environments, a model canvas was built to support the planning and management of *software analytics* activities based on the patterns presented in the Chapter 4.

6 SOFTWARE ANALYTICS CANVAS EVALUATION

This chapter presents the evaluation cycle of the first version of SA Canvas presented in previous chapter. One of the question in this study was to verify if the proposed artifact supports the cognitive activities related to the planning and management of *software analytics* activities. The results of the assessment and suggestions from study participants provided us with insights for improving the design of the canvas and its components. The second version of SA Canvas is presented in Chapter 7.

Section 6.1 introduces the study design that encompasses the objectives, planning, and execution steps of this evaluation cycle. Section 6.2 describes the method used for cognitive activities analysis. Section 6.3 presents the results, answering how does the SA Canvas support the cognitive activities and what the participants' perceptions about the usefulness and ease-of-use of the SA Canvas. And, Section 6.4 discusses the study limitations and threats to validity.

6.1 Study design

With the aim of evaluating the first version of SA Canvas artifact and improving its characteristics, we have carried out a formative evaluation. The purpose of formative evaluations is to help improve the outcomes of the process under evaluation. According to Venable et al. (VENABLE et al., 2016), formative evaluations can provide a basis for successful action in improving the characteristics of an artifact from empirically-based interpretations.

6.1.1 Research questions

The main goal of the formative study was to investigate how the SA Canvas supports the cognitive activities of practitioners related to *software analytics*. Additionally, we gathered participants' perceptions about the usefulness and ease-of-use of the artifact and their enhancement suggestions. The following research-questions were the main focus of the artifact evaluation:

- RQ1) How does the SA Canvas support the cognitive activities of practitioners related to *software analytics*?
  - What are the actions that characterize the participants' interaction strategies?
  - How are resources combined to guide participants' actions?
  - What interaction patterns emerge from iterations over time?

- RQ2) What are the participants' perceptions about the usefulness and ease-of-use of the SA Canvas?

- RQ3) What characteristics can be improved in the design of the SA Canvas?

To answer RQ1, we first analyzed what were the actions that characterized each of the participants' strategies throughout the planning and management of *software analytics* project. Second, we analyzed the sequencing of interactions in the temporal view to map the interaction strategy, components of the canvas and other artifacts. And then, we analyzed the shift between interaction strategies along the iterations by group.

To answer RQ2, we collected participants' perceptions about the usefulness and ease of use of the artifact after iterations with the canvas, using a questionnaire based on the Technology Acceptance Model (TAM) (DAVIS, 1989), an intention-based model widely used to explain or predict user acceptance of computer technology. And, to answer RQ3, we call the same subjects for a participatory design session to suggest ideas for possible artifact enhancements.

6.1.2 Participants

The study participants were six individuals, master and doctoral students who were enrolled in the course of Agile Projects at the National Institute for Space Research (INPE). Table 6.1 shows the participants' characterization in terms of background, experience in developing software, and their practice on agile methods.

Table 6.1 - Participants characterization.

Pairs	Subjects	Background	Software Development[1]	Agile Methods [2]
G1	P1	web development	6 to 10	little
	P2	software architecture	16 to 20	reasonable
G2	P3	systems analysis	3 to 5	none
	P4	systems analysis	11 to 15	little
G3	P5	computer programming	3 to 5	little
	P6	systems analysis	16 to 20	reasonable

[1] experience in years
[2] practical work

SOURCE: Prepared by the author.

Participants had at least three years of professional experience in the software industry and collaborative work. Moreover, four of them also were familiar with the application domain involving sensors and spatial data. Only one of them had no experience with agile methods. We decided to conduct the study in pairs to stimulate the conversation between the participants. Considering the participants' experience, we seek to form balanced pairs. The observational study sessions were carried out in the Laboratory of Computation and Applied Mathematics.

6.1.3 Study planning and execution

In total, five meetings were held with the participants, during the period from November 6 to December 11, 2018. The first four meetings were dedicated to observational study on cognitive activities (RQ1) and collecting the participants' perceptions of the artifact's usefulness and ease of use (RQ2). And, the fifth meeting was dedicated to a participatory design session to collect suggestions for canvas improvements (RQ3).

6.1.3.1 Observational study

In the observational study, the pair of participants were invited to plan and manage a *software analytics* project using the first version of SA Canvas. This study was based on the real architecture of the EMBRACE's system [1] and real issues reported by the development team. Due to the type of evaluation we wanted to do, we opted for a controlled study that would allow us to capture details of the participants' interaction with our canvas.

Materials. Previously, we prepared a document describing the space weather application and some real concerns extracted from meetings with the EMBRACE's development team. Such document was used as an input for the observational study. Also, a SA Canvas was prepared using whiteboards to support the pairs of participants during the study. As a additional artifacts, we prepared a list of possible data sources, and a document based on recommendations for use of SA Canvas as presented in the Chapter 5 - Section 5.5.1.

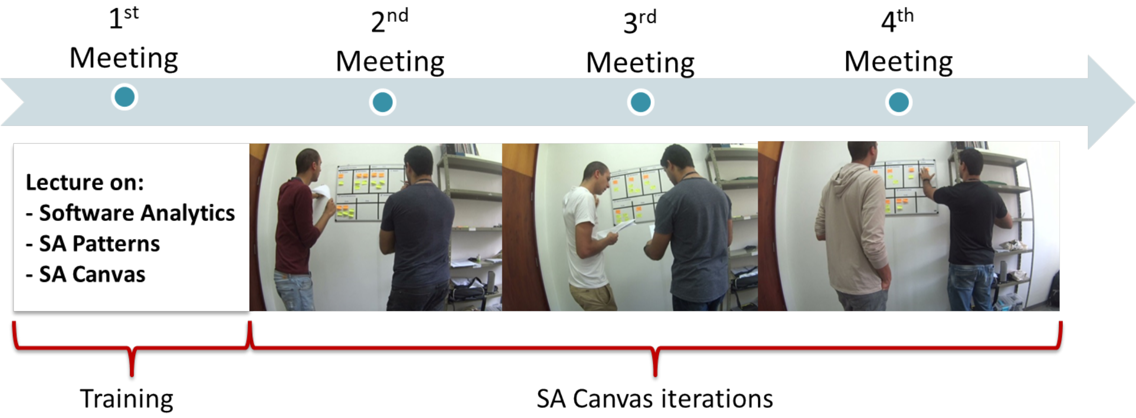
Pilot study. First of all, we carried out a pilot study with two researchers from the computer lab to evaluate their iteration with the SA Canvas reproduced on the whiteboard and the other materials – case study description, list of possible data sources, and canvas guidelines.

[1] Site: http://www2.inpe.br/climaespacial/portal/pt/

From the pilot study, we established that (i) pairs should have to raise at least two issues of *software analytics*; and that (ii) a warm-up exercise together with all participants should be carried out before they begin planning their projects.

Study execution. To verify how cognitive activities related to *software analytics* are supported by the proposed artifact, an observational study was undertaken during four weekly meetings with participants, as presented in Figure 6.1.

Figure 6.1 - Observational study meetings over time.



SOURCE: Prepared by the author.

1st Meeting) At the first meeting, we introduced concepts about *software analytics*, presented an overview of SA pattern, and presented SA Canvas using the practical example similar to that mentioned in the previous chapter (Section 5.5.3). For obtaining more information before the next meeting, we asked the participants the reading two articles on *software analytics* patterns (CHOMA et al., 2017) (CHOMA et al., 2018) and one article presenting the architectural model for the collection, processing, and visualization of space weather information used in EMBRACE (SANT'ANNA et al., 2014). After this first meeting, participants read and signed an informed consent form to participate in the study (see Appendix C).

2nd Meeting) At the second meeting, we invited the participants to a practical exercise to understand how they should use the canvas. One of the participants provided us with a real example of his own work. From this example, we helped them to plan *software analytics* activities on the board using sticky notes.

This activity took approximately one hour. After warming up, a pair of participants at a time should fill out four blocks of the canvas (Key Issues, Data Sources, Data Gathering, and Analytics Implementation) based on information written in the document on the EMBRACE case study and the support material, i.e., list of possible data sources and the document with recommendations for the use of canvas named "canvas tutorial" (see Appendix D). Additionally, we encourage teams to use different color sticks for different key issues. Each pair took from 25 to 40 minutes to fill the four blocks with at least two key issues. The sessions were recorded for future analyzes, and one of the researchers observed the activities in silence, taking notes on the interaction of participants with canvas and other artifacts. After this iteration, we prepared a report with (i) feedback about how they had used the canvas including corrective actions, and (ii) fictitious results to encourage participants to update the canvas and move the evaluation process forward. Also, we have introduced some insights mixed into the text to be extracted by them. Because each team raised different key issues, we then prepared three different documents for each of them.

3rd Meeting) The report's information was used as input for the second iteration when the team should adjust some fill-in mistakes pointed out by researchers, update the canvas with the tasks done, proceed in the planning their activities for the next iteration, and fill in the remaining canvas blocks from the insights. As in the first iteration, the sessions were recorded and observed by one of the researchers, again in the non-participatory way. The sessions took from 40 to 60 minutes. From the results of this iteration, we prepared a new report for each team with the feedback on the use of the canvas, other fictitious results about the evolution of the project, and new insights mixed into the text.

4th Meeting) As in the previous iteration, from the researchers' report, the teams should adjust some fill-in mistakes pointed out by researchers, update the canvas with the tasks done, proceed in the planning their activities for the next iteration, and fill in the remaining canvas blocks from the insights. The sessions took on average 40 minutes, and just like the previous ones, they were recorded and observed by one of the researchers.

In summary, three iterations took place in which each group made use of the SA Canvas to planning and managing their tasks, totaling 371 minutes of videos recorded for further analysis. Table 6.2 shows the time spent in each of the nine meetings held by the pairs.

Table 6.2 - Time spent in meetings.

Pairs	1° Iteration	2° Iteration	3° Iteration
G1	24 min	36 min	37 min
G2	44 min	50 min	43 min
G3	39 min	60 min	38 min

SOURCE: Prepared by the author.

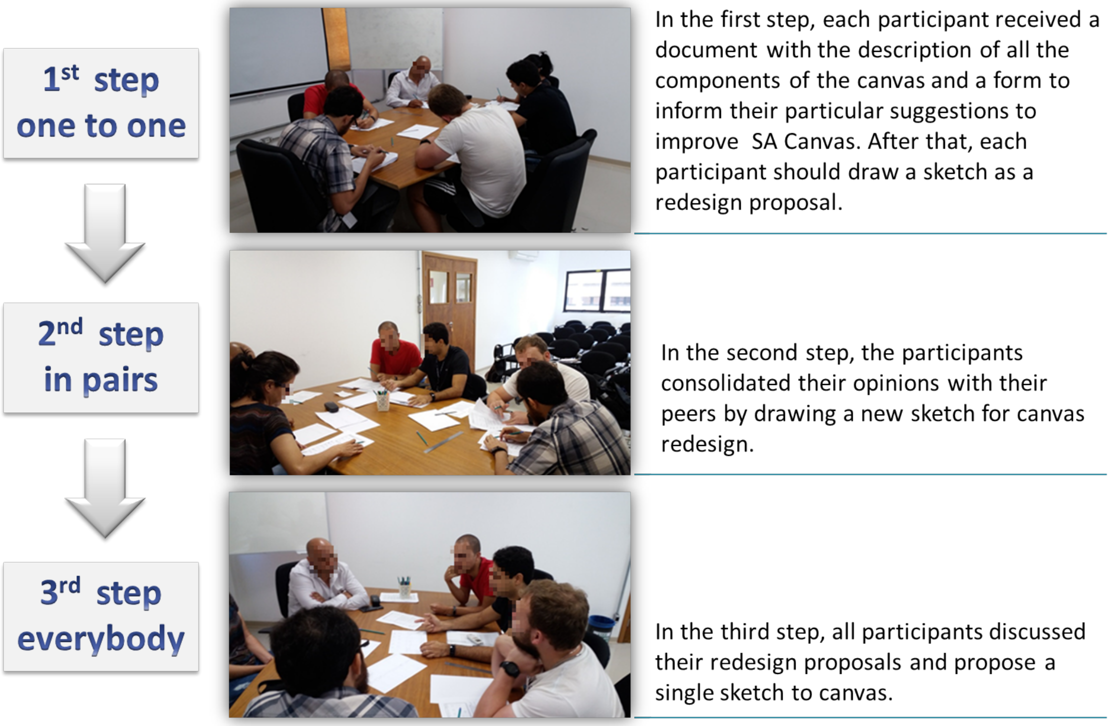
6.1.3.2 Participants perception questionnaire

The participants' individual perceptions of usefulness and ease of use on proposed artifact were collected at the same day of the 4th meeting, right after the hands-on experience. A questionnaire based on Technology Acceptance Model (TAM) (DAVIS, 1989) was used as collect instrument to *perceived usefulness* (PU) and *perceived ease of use* (PEU). PU refers to "the degree to which a person believes that using a particular system would enhance his or her job performance"; and PEU refers to "the degree to which a person believes that using a particular system would be free of effort". Both variables were measured through a multiple-choice questions, using a 6-point Likert scale – from "Completely Disagree" to "Completely Agree".

6.1.3.3 Participatory design

To investigate what how SA Canvas could be improved, we conducted a participatory design session on a fifth date with the same participants from the observational study. The method used to conduct the redesign section was divided into three steps. I and the other researcher participated in the session as observers making notes of relevant information. On average, each step lasted 30 minutes. In the first step, each participant received a document with the description of all the components of the SA canvas and a form requesting suggestions for design enhancement of the proposed artifact. Participants were asked to individually respond to whether they suggested any changes to the canvas (for example: extinguishing a component, creating a new component, or merging one component with one another), and how it could be improved. Figure 6.2 summarizes the methodology used to conduct the participatory design session.

Figure 6.2 - Participatory design session.



1st step one to one

In the first step, each participant received a document with the description of all the components of the canvas and a form to inform their particular suggestions to improve SA Canvas. After that, each participant should draw a sketch as a redesign proposal.

2nd step in pairs

In the second step, the participants consolidated their opinions with their peers by drawing a new sketch for canvas redesign.

3rd step everybody

In the third step, all participants discussed their redesign proposals and propose a single sketch to canvas.

SOURCE: Prepared by the author.

6.2 Method of analysis

To assess how the SA Canvas supports the **cognitive activities** related to *software analytics*, we recorded on video the nine observation sessions where SA Canvas was used by three pairs of participants. For data analysis, we considered only the data that were observable through the actions and conversations between the participants. Then, to explore cognitive activities captured during planning activities, an approach for analysis based on Resource Model (RM) (WRIGHT et al., 2000) and Sequential Analysis (SA) (BAKEMAN; GOTTMAN, 1997) was adopted, as introduced in Chapter 3, Section 3.2.2.

In this method of analysis, RM was used as a framework to the analysis of software practitioners' interactions with external resources by mapping the participants' actions to *interaction strategies* prescribed in this model – i.e, plan construction (PC), plan following (PF), goal matching (GM), and history-based choice (HC) (WRIGHT et al., 2000).

Note that, external resources in other software engineering activities could be any objects, artifacts, and support materials – e.g., diagrams, reports, sketches, prototypes, patterns, guidelines, cards describing the software features, physical or digital boards on which practitioners take to plan their tasks and to monitor their work in progress. In our study, we considered various resources available in the study setting to support participants during their tasks. However, our main interest was focused on the use of SA Canvas. After interaction strategies characterization, we undertaken a sequential analysis using LSA (BAKEMAN; GOTTMAN, 1997) to identify patterns in *sequences of interaction strategies*.

In summary, the proposal for coding and analyzing the observed interactions consisted of five steps:

i. Describing the events sequentially from observing interactions between software practitioners and available resources over time;

ii. Categorizing the interactions events following a coding scheme based on the observed actions;

iii. Associating of the emergent actions to the *interaction strategies* defined in the RM (WRIGHT et al., 2000);

iv. Creating of graphical with a temporal view to identifying resources combined during participants' actions; and

v. Analysis of the interaction strategies by applying lag-sequential analysis using Bakeman & Gottman's formula (BAKEMAN; GOTTMAN, 1997), where a transition from one state to another is considered significant if the z-score was higher than 1.96. Then, state diagrams are used to show the patterns of participant's transitions from one interaction strategy to another across time.

6.3 Results

This section presents the results of the observational study on cognitive activities and findings on participants' perception regarding canvas usage. And, the findings from the participatory design session will be presented in Chapter 7, along with the second version of the canvas.

6.3.1 Cognitive activities analysis

To verify how the SA Canvas supports the cognitive activities of participants during the planning and management of *software analytics* issues, the six steps of the coding and analysis method presented in the previous section were applied.

Events description (i): the pairs interactions with the artifact under evaluation and other available resources were described as a set of events over time from the transcription of videos. Table 6.3 shows the number of sequential events described from nine iterations.

Table 6.3 - Number of events per iteration.

Pairs	1° Iteration	2° Iteration	3° Iteration
G1	104	92	77
G2	122	87	72
G3	112	111	69

SOURCE: Prepared by the author.

It is worth noting that events could involve pair interactions with more than one available resource. The main resources observed include the seven components draw on the canvas board (CB): Key Issues (KI), Data Sources (DS), Data Gathering (DG), Insights (IS), Quality Thresholds (QT), Analytics Implementation (AI), and Incremental Goals (IG). The other secondary resources also observed and included as support artifacts were: scenario EMBRACE case (EC), list of possible data sources (DS), canvas' tutorial (CT), sticky notes (PT), two reports on tasks and insights (R1 and R2), feedback on the use of canvas (F1 and F2), and complementary material such as two articles on SA patterns and the article on the architectural model used in EMBRACE.

Emergent actions (ii): each event should be categorized following a coding scheme based on the observed actions, which are coded as a behavioral unit. The coding scheme could be established before data gathering. However, our coding occurred during the analysis by following an inductive approach. This approach is usually undertaken when a research question or interaction context is entirely novel, such that published coding schemes cannot be applied to the data at hand (LEHMANN-WILLENBROCK; ALLEN, 2018). The events were categorized into 27 emergent actions. The list of observable actions along with a sample of the description of some events is presented in Table 6.4.

Table 6.4 - List of emergent actions.

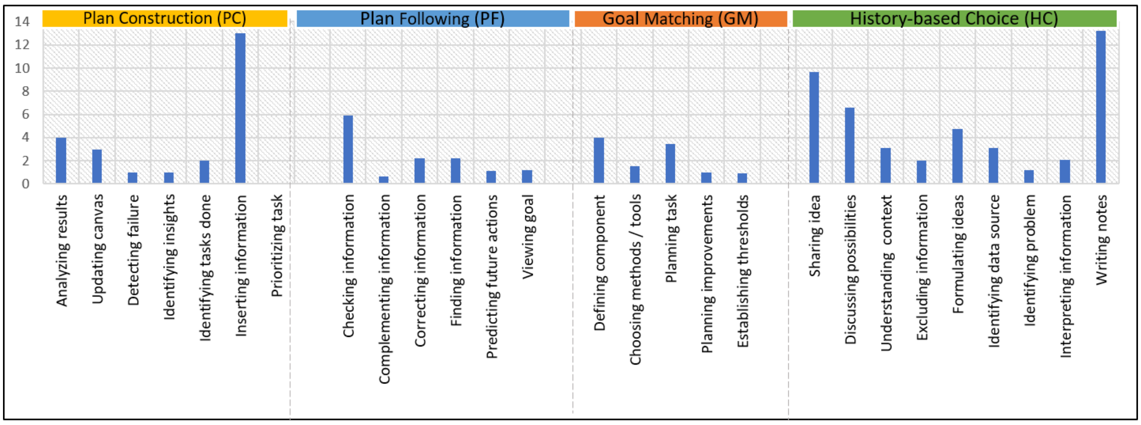
Actions	Examples of events description
Analyzing results	P2 analyzes insights on the board across report information [G3-i3]
Checking information	P2 checks all data sources that have been defined by looking at the canvas board [G3-i1]
Choosing methods/tools	P2 suggests Bugzilla as the bug-tracking tool for monitoring the legacy systems [G2-i1]
Complementing information	P1 adds information in the sticky note to make clearer the target of the task [G1-i2]
Correcting information	P1 right the early note of insight referring issue2 for “sensor reliability” [G2-i3]
Defining component	P2 looking for the definition of the component “insights” in the canvas tutorial [G1-i2]
Detecting failure	P1 realized that they used sticky notes of the same color for the three issues [G1-i1]
Discussing possibilities	P2 comments that "it depends on how the system uses the database to register", and mentions the possibility of access by login. P1 disagree with the idea by explaining that many sites do not use login to access [G3-i1]
Establishing thresholds	P1 and P2 discuss values for the thresholds. P1 suggests an error rate at 30% [G2-i3]
Excluding information	P1 excludes the two insights from canvas board that no longer make sense at that moment [G3-i2]
Finding information	P2 asks whether there is a real need for refactoring. P1 search something about it in the report [G3-i3]
Formulating ideas	P1 elaborates a task to create tests, first verifying in the report how data is accessed [G2-i3]
Identifying data source	P1 identifies the need to define a data source for data gathering about “access records” [G2-i1]
Identifying insights	P1 and P2 identify an insight related to the database from the findings in the report [G3-i2]
Identifying problem	P1 and P2 identify the need to verify access of legacy systems when reading the document on the case [G2-i1]
Identifying tasks done	P2 points the task1 about the verification of database on the canvas that had already been done [G3-i3]
Inserting information	P1 inserts the note of a task for identifying the modules of legacy system on the analytics to-do session [G3-i3]
Interpreting information	P1 and P2 discuss the report content to understand the cause of the access record problem [G3-i2]
Planning improvements	P1 re-evaluates an action for testing and exchanging the faulty sensors [G1-i3]
Planning task	P1 and P2 plan a task to meet the needs of the developers noted in the report [G2-i2]
Predicting future actions	P2 recognizes that the mentioned task could be a solution possible if a given problem was confirmed [G2-i2]
Prioritizing task	P1 and P2 agree to prioritize two activities before proceeding with the analysis [G1-i3]
Sharing idea	P1 suggests measuring the size of files right after processing them, and P2 agrees with the idea [G3-i1]
Understanding context	P2 reads aloud the case study excerpt related to the quality of data to discover how this is handled [G2-i2]
Updating canvas	P1 moves task1 referring to issue1 from “to-do” to “done” on analytics implementation session [G3-i3]
Viewing goal	P1 searches on the board what is the “ key issues” related to the findings pointed out by P2 [G3-i2]
Writing notes	P1 writes in a sticky note “google analytics” using the corresponding color of the “key issue” [G1-i1]

SOURCE: Prepared by the author.

The top five actions found were *inserting information* [in the canvas board], *writing notes* [using stick-notes], *sharing ideas* [with partner], *discussing possibilities*[with partner], and *checking information*[contained in the canvas board, reports, or other documents]. These actions are highlighted in blue, in the Table 6.4.

Interaction strategy (iii): after a broad analysis of participants' interactions with the artifact under evaluation and other available resources, the emergent actions were associated with the interaction strategies of the Resource Model (WRIGHT et al., 2000) – i.e., plan following, plan construction, goal matching, and history-based choice. In this step, the purpose was to verify what types of actions characterized the interaction strategies. Figure 6.3 displays the bar graphs with the average frequency of emergent actions over three iterations grouped by interaction strategies.

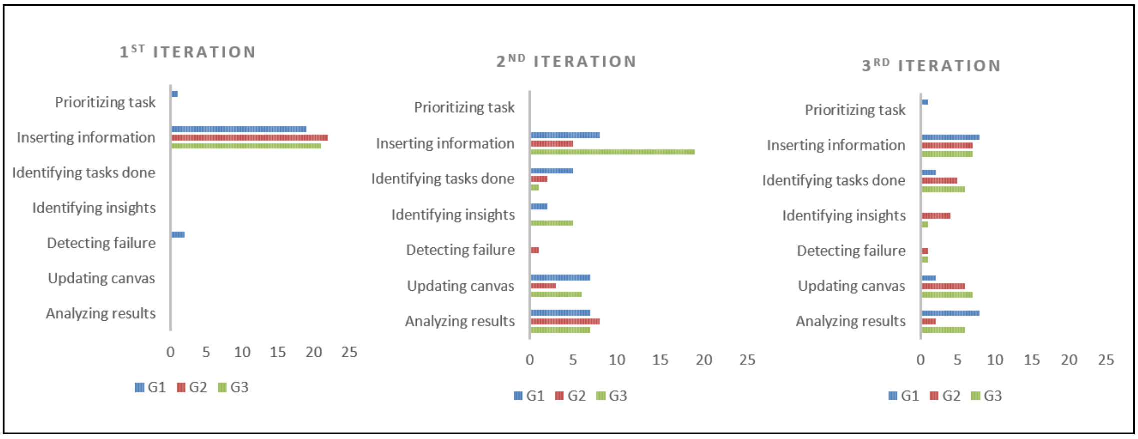
Figure 6.3 - Number of actions grouped by interaction strategy.



SOURCE: Prepared by the author.

Figures 6.4 to 6.7 present the bar graphs grouped by interaction strategies with the number of emergent actions accumulate during the iterations of each group. The actions of *plan construction* strategy are focused on canvas construction and maintenance, for instance, when it involves the inserting of information since the first iteration, and updating of information from the second iteration (Figure 6.4).

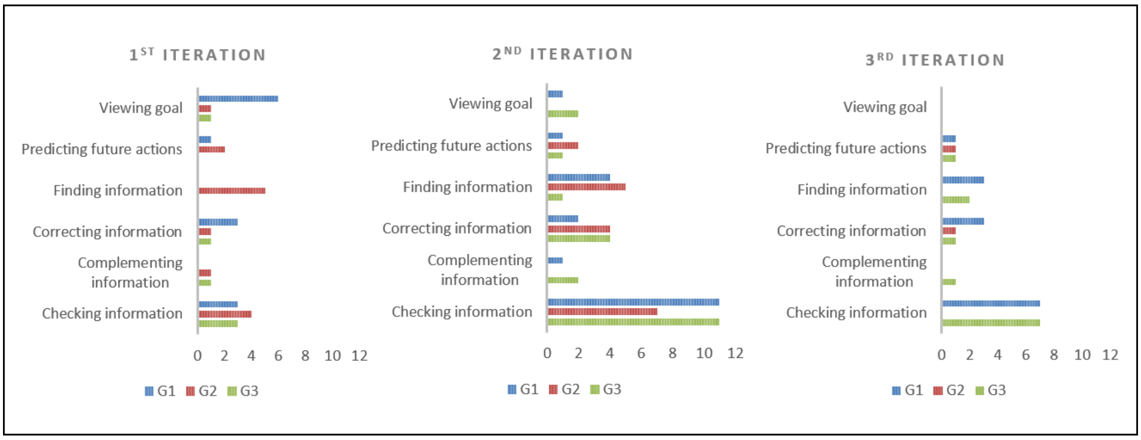
Figure 6.4 - Plan construction strategy.



SOURCE: Prepared by the author.

In the *plan following* strategy, the action “checking information” usually occurs before action of “updating canvas” that is part of maintenance activity (Figure 6.5). However, in the second iteration, the participants had to check information more times to fix some mistakes made at the previous iteration as pointed out in the researchers’ report. Moreover, over the three iterations, participants needed to know more about a specific component before entering or fixing information related to it.

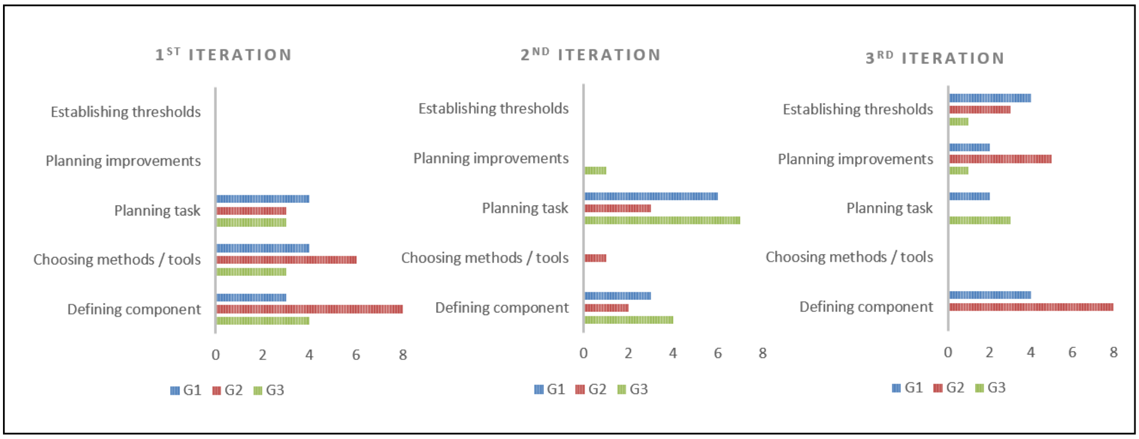
Figure 6.5 - Plan following strategy.



SOURCE: Prepared by the author.

Figure 6.6 presents that most of the actions of *goal matching* strategy is directly related to the components of the canvas. That is why some actions were only carried out at the beginning of the study (e.g., choosing methods and tools), whereas some actions are only performed from the second iteration when the participants received new data, fictitious results, and insights to proceed with filling the canvas board (e.g., planning improvements and establishing thresholds).

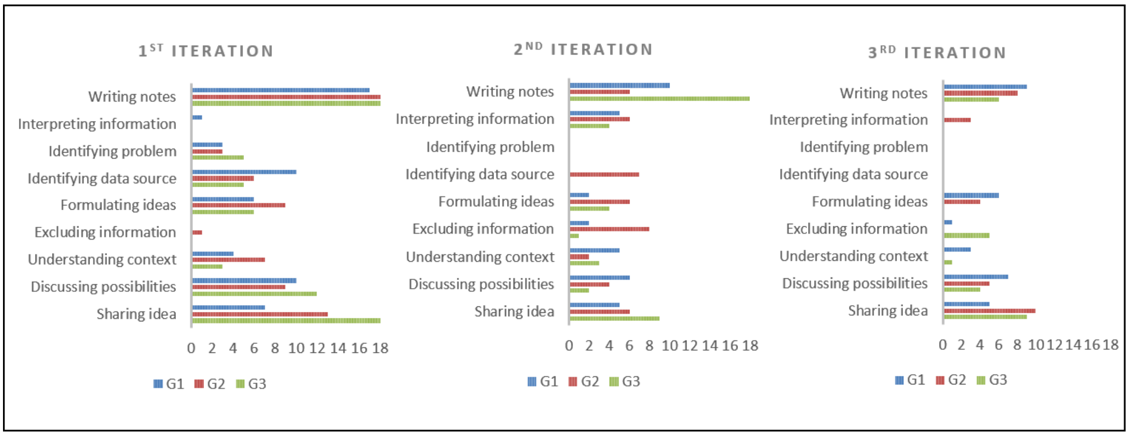
Figure 6.6 - Goal matching strategy.



SOURCE: Prepared by the author.

In the *history-based choice* strategy, the most of actions depend on the participants' reflection on previously made decisions or pre-established conditions (Figure 6.7).

Figure 6.7 - History-based choice strategy.



SOURCE: Prepared by the author.

Part of the information needed for decision making is in the minds of the participants, another part is distributed within the supporting artifacts or is part of the interaction history. For example when they write a note after formulating the content in their minds, after discussing possibilities and sharing ideas with their partners or consulting the history contained in the artifacts. Note that, the identifying of the problem occurs only in the first iteration when the participants define the key issues based on the case study history.

Observations over time (iv): in this step, nine graphs in the format of timelines were generated based on one-minute intervals to obtain a visual representation of the participants' interactions with components of canvas and other available resources. In the same graphs, we included the interaction strategies adopted by the participants. The timelines of all groups are presented in Appendix E.

The graphical visualization of the observations in a temporal view should facilitate the researcher's analysis toward understanding how resources were combined to guide the practitioners' actions. When we analyzed the actions and sequencing of interactions over time, we were able to identify, for instance, that group G2 in the first and third iterations had more difficulties in understanding some components of the canvas (e.g., Data Sources, Data Gathering, and Incremental Goals) since they more frequently consulted the document with canvas guidelines (see Appendix E).

Sequential analysis (v): with the purpose of uncovering strategy exchange patterns during participants' interactions with components of canvas and other available resources, we draw nine diagrams using weighted directed graphs, as presented in Figure 6.8. The diagrams provide a visual way for analysis of participants' all sequential moves by switching from one strategy to another in each iteration.

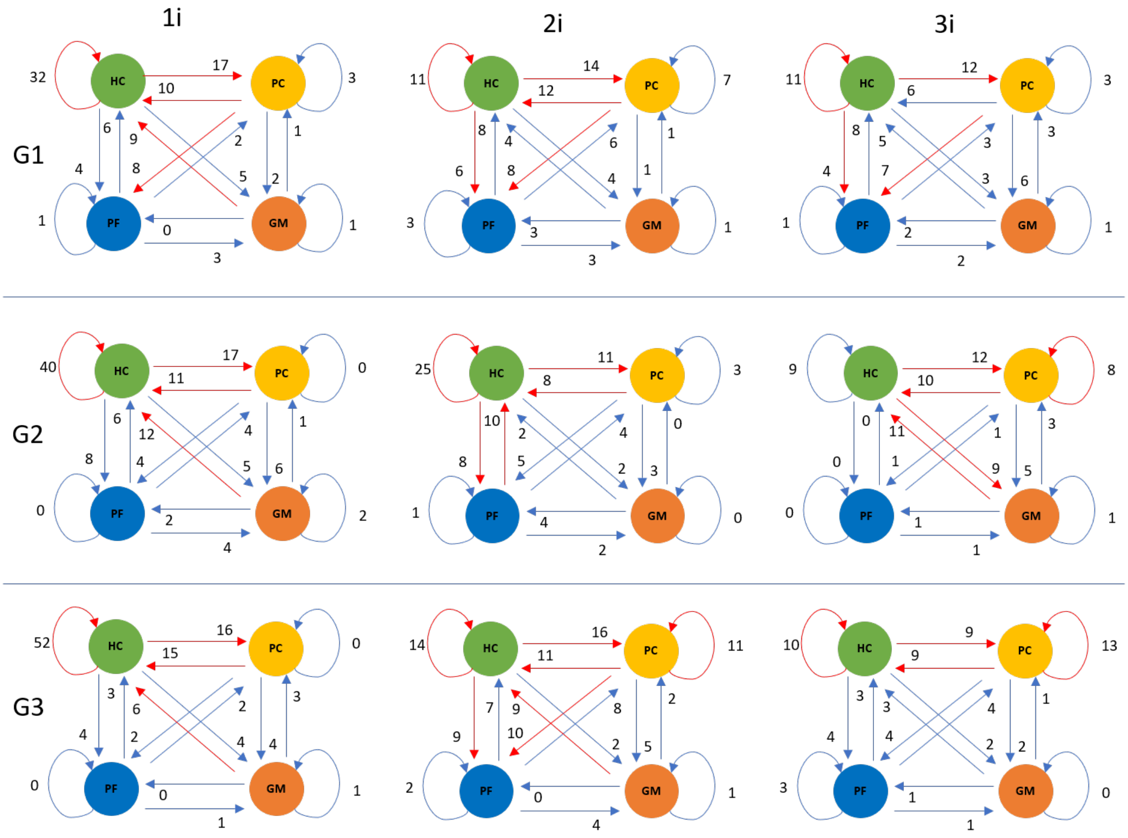
With the aim of identifying significant relationships between such moves, the lag-sequential analysis (LSA) (BAKEMAN; GOTTMAN, 1997) was applied. Sequential analysis generally is employed (a) to discover probabilistic patterns in the stream of code events, or (b) assess the effect of contextual or explanatory variables on the sequential structure of interactions (BAKEMAN; GOTTMAN, 1997). To examine whether the observed transition probabilities were statistically significant, we used software developed by CoSci Research Group[2], which uses the Bakeman & Gottman's formula (BAKEMAN; GOTTMAN, 1997):

[2] Available at: https://cosci.tw/lsaPage

$$z_{ij} = \frac{x_{ij} - m_{ij}}{\sqrt{m_{ij} * (1 - x_{i+}/N)(1 - x_{+j}/N)}}$$

where x_{ij} is the observed number and $m_{ij} = x_{i+}x_{+j}/N$ is the expected number of transitions from event i to event j, with x_{i+} being the total observed counts of the *i*-th row, x_{+j} is being the total observed counts for the *j*-th column, and N being the total number of records in the table. Since this formula uses the z-statistic based on the normal distribution, values higher than 1.96 (or lower than 1.96) can be considered statistically significant.

Figure 6.8 - Interaction strategy diagram.

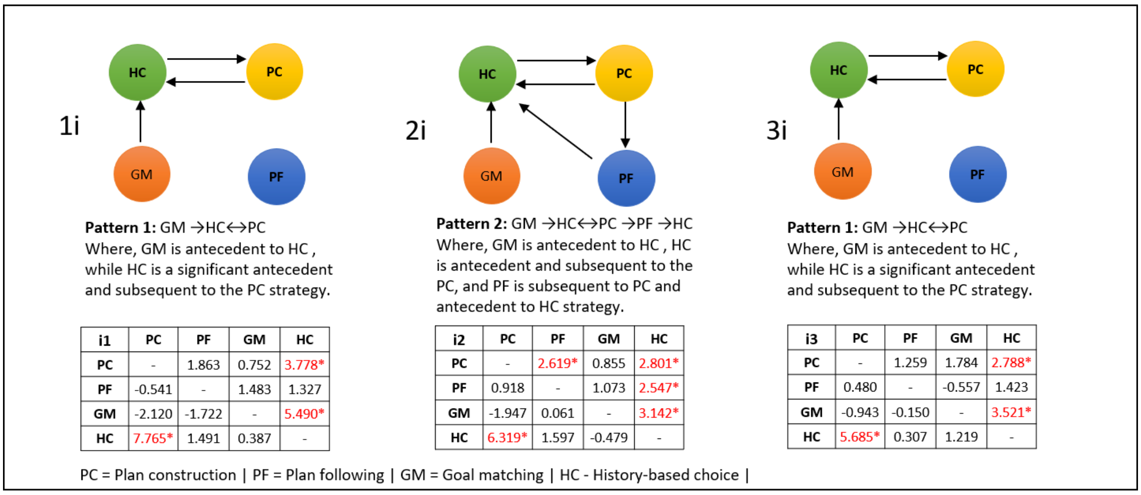


The circles in the diagram depict different strategies. and the arrows display the transitional frequency between them. The arrows in red highlight the most frequent moves. PC = Plan construction, PF = Plan following, GM = Goal matching, HC = History-based choice.

SOURCE: Prepared by the author.

Figure 6.9 presents the diagrams resulting along with LSA matrix for the moves. LSA is useful for indicating the degree of confidence with which it can be stated that a given event influences the occurrence of another (OLSON et al., 1994). The unit of analysis in the sequential analysis is a two-event sequence recorded across some period of time or across conditions or contexts. In this study, the unit of analysis is the sequence of interaction strategies observed in each iteration. For this analysis, all groups are considered. Moreover, self-loops are not computed, for example, when a history-based choice move (HC) is subsequent to another HC move.

Figure 6.9 - Significant relationships between iterations strategies.



Pattern 1: GM →HC↔PC
Where, GM is antecedent to HC , while HC is a significant antecedent and subsequent to the PC strategy.

i1	PC	PF	GM	HC
PC	-	1.863	0.752	3.778*
PF	-0.541	-	1.483	1.327
GM	-2.120	-1.722	-	5.490*
HC	7.765*	1.491	0.387	-

Pattern 2: GM →HC↔PC →PF →HC
Where, GM is antecedent to HC , HC is antecedent and subsequent to the PC, and PF is subsequent to PC and antecedent to HC strategy.

i2	PC	PF	GM	HC
PC	-	2.619*	0.855	2.801*
PF	0.918	-	1.073	2.547*
GM	-1.947	0.061	-	3.142*
HC	6.319*	1.597	-0.479	-

Pattern 1: GM →HC↔PC
Where, GM is antecedent to HC , while HC is a significant antecedent and subsequent to the PC strategy.

i3	PC	PF	GM	HC
PC	-	1.259	1.784	2.788*
PF	0.480	-	-0.557	1.423
GM	-0.943	-0.150	-	3.521*
HC	5.685*	0.307	1.219	-

PC = Plan construction | PF = Plan following | GM = Goal matching | HC - History-based choice |

The circles in the diagram depict different strategies, and the arrows display only significant relationships, according to values highlighted in red in the tables below each diagram.

SOURCE: Prepared by the author.

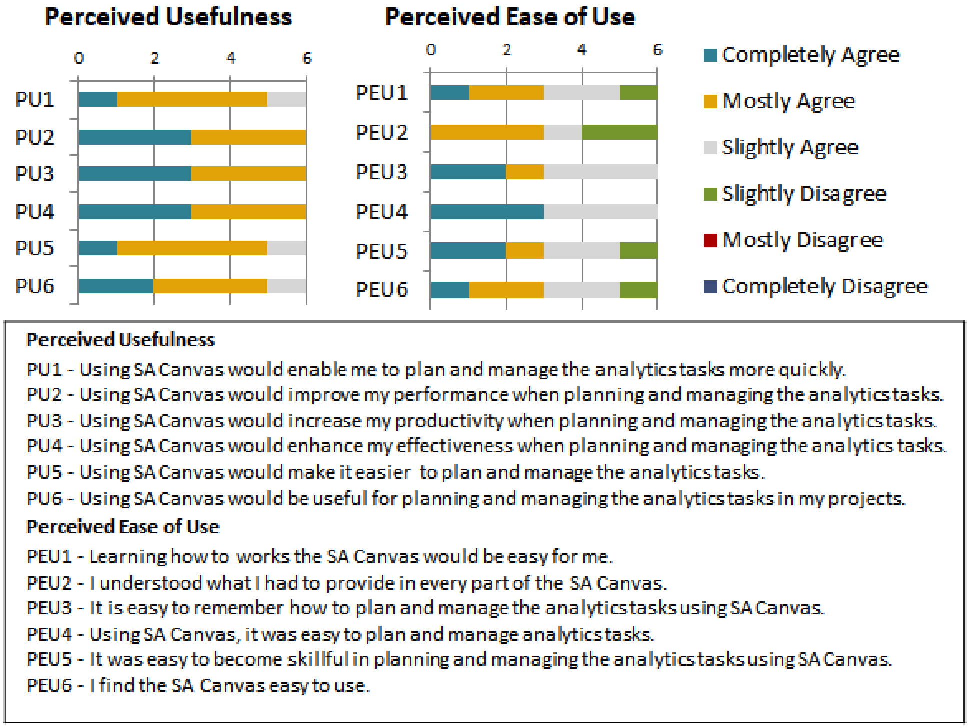
As presented in Figure 6.9 (at the bottom), pairs of events are organized into a contingency table where, by convention, the antecedent occupies the rows, and the subsequent move occupies the columns. Regarding interaction patterns, we found one pattern in the first and third iteration - i.e., GM → HC ↔ PC. While, in the second iteration, the emerged pattern was GM → HC ↔ PC → PF → HC. When interpreting the first pattern characterized in the first and third iterations, we can argue that it describes an expected flow since the GM strategy is focused on the actions that refer to canvas filling. These actions trigger movements to solve the problem, including reasoning, discussion, sharing, and the formulation of ideas (HC).

Finally, the cycle closes with the PC strategy, where participants complete their tasks when inserting information on the canvas. Eventually, there are movements in which the PC strategy precedes HC. This fact is especially true when individuals need to analyze some results before accomplishing a task. In the second pattern, we can noted that there were more significant movements in addition to the movements pointed out in the previous pattern. This pattern can be explained because, in this iteration, the participants received feedback on the use of the canvas and had the additional task of making some adjustments. At this point, we were able to verify what were the main difficulties of the participants.

6.3.2 Usefulness and ease-of-use evaluation

After three iterations with the canvas, the participants were able to evaluate the usefulness and ease of use of the artifact (RQ2). The data collected from TAM-based questionnaire were analyzed using descriptive statistics. The Figure 6.10 displays the bar graphs that summarize the responses of the participants (at the top), and the questions elaborated for the questionnaire using Likert scale (at the bottom).

Figure 6.10 - Perceived usefulness and ease-of-use.



Perceived Usefulness

PU1 - Using SA Canvas would enable me to plan and manage the analytics tasks more quickly.
PU2 - Using SA Canvas would improve my performance when planning and managing the analytics tasks.
PU3 - Using SA Canvas would increase my productivity when planning and managing the analytics tasks.
PU4 - Using SA Canvas would enhance my effectiveness when planning and managing the analytics tasks.
PU5 - Using SA Canvas would make it easier to plan and manage the analytics tasks.
PU6 - Using SA Canvas would be useful for planning and managing the analytics tasks in my projects.

Perceived Ease of Use

PEU1 - Learning how to works the SA Canvas would be easy for me.
PEU2 - I understood what I had to provide in every part of the SA Canvas.
PEU3 - It is easy to remember how to plan and manage the analytics tasks using SA Canvas.
PEU4 - Using SA Canvas, it was easy to plan and manage analytics tasks.
PEU5 - It was easy to become skillful in planning and managing the analytics tasks using SA Canvas.
PEU6 - I find the SA Canvas easy to use.

SOURCE: Prepared by the author.

When analyzing participants' responses as to perceptions of usefulness and ease of use of the proposed artifact, we noted that to some degree all participants agree that the SA Canvas is usefulness (Figure 6.10). However, there was some disagreement concerning the ease of use, according to the graph at right. Some participants do not agree that it was easy to learn and understand what should be done at certain times (PEU1 and PEU2). The learning curve certainly will have an impact on the user's ability to handle the artifact (PEU5). Furthermore, users tend to be more critical as to the usefulness of the artifact when its use is not effortless.

6.4 Limitations and threats to validity

The empirical studies are commonly subject to limitations and threats to validity. In this section, we report possible threats to the validity of our observational study in terms of construct, internal, external, conclusion validity, and reliability (PETERSEN; GENCEL, 2013).

Construct validity refers to the measures selected to answer research questions (PETERSEN; GENCEL, 2013). All sessions of the observational study were video recorded. For the codification of the participants' cognitive activities identified during the analysis of the videos, we used a set of interaction strategies prescribed from the Resources Model (WRIGHT et al., 2000). Furthermore, to identify significant patterns we use the Sequential Analysis concepts (BAKEMAN; GOTTMAN, 1997). In addition, in order to guarantee the data quality, we conducted a warm-up with the participants in advance to eliminate any misunderstanding about how the canvas components should be filled out. To collect participants' perceptions, we use TAM - a well-established tool in the area of software engineering. And, to collect suggestions for improvements we use the DP method which is widely used in the area of user centered design.

Internal validity is about investigating whether one factor affects an investigated factor (PETERSEN; GENCEL, 2013). The study protocol was previously planned and refined through a pilot test. The study was divided into weekly meetings to characterize the time-box of iterations that normally occur in agile environments. The fact that the teams need to remember their past actions before evolving in filling the canvas was a way to evaluate the usefulness of the artifact.

External validity is concerned to generalize the findings considering variations of individuals, settings, treatments, and outcomes (PETERSEN; GENCEL, 2013). Our study was carried out with pairs of individuals who were experienced in different areas of software development and with different backgrounds. Also, one of the pairs had expertise in the domain of the case studied. The study was based on issues relevant to the real world. Although the starting point focused on a specific case, each pair identified different issues and followed a different path to resolve their issues.

Conclusion validity is concerning how researchers derive their conclusions from the relationship between study variables (PETERSEN; GENCEL, 2013). To mitigate the researcher's bias when determining significant movements using absolute frequency, we used the lag-sequential analysis (LSA) to test whether the observed transition probabilities were statistically significant.

Reliability is concerning to verify whether data analysis depends on a specific researcher (PETERSEN; GENCEL, 2013). A single person performed the analysis and coding which after were refined and discussed with three other senior researchers; and besides, the analysis protocol was previously discussed and validated with two other researchers. Furthermore, all interactions during the observational study were video-recorded that enabled the researcher to double-check the data to confirm described events, coded actions, and the movements of the participants.

6.5 Chapter summary

This chapter presents the results of the evaluation of the first version of SA Canvas. The evaluation step consisted of an observational study to assess the use of the artifact in support of cognitive activities, the collection of information on the usefulness and ease of use of the artifact from the point of view of the participants, and a participatory design session to collect suggestions improvements. In the next chapter, the points of improvement suggested participants during the participatory design session are listed with the second version of the canvas.

7 SOFTWARE ANALYTICS CANVAS REDESIGN

This chapter presents the second version of SA Canvas redesigned from evaluation study outcomes reported in the previous chapter and the study participants' suggestions for improving the canvas which are introduced in this chapter. The initiatives to evaluate the second version of the canvas in practice are also presented in this chapter.

Section 7.1 lists the participatory design findings. Section 7.2 introduces the new template of canvas and describes the changes carried out based on evaluation study outcomes. Section 7.3 covers the new round of evaluations of the second version by collecting feedback from software practitioners.

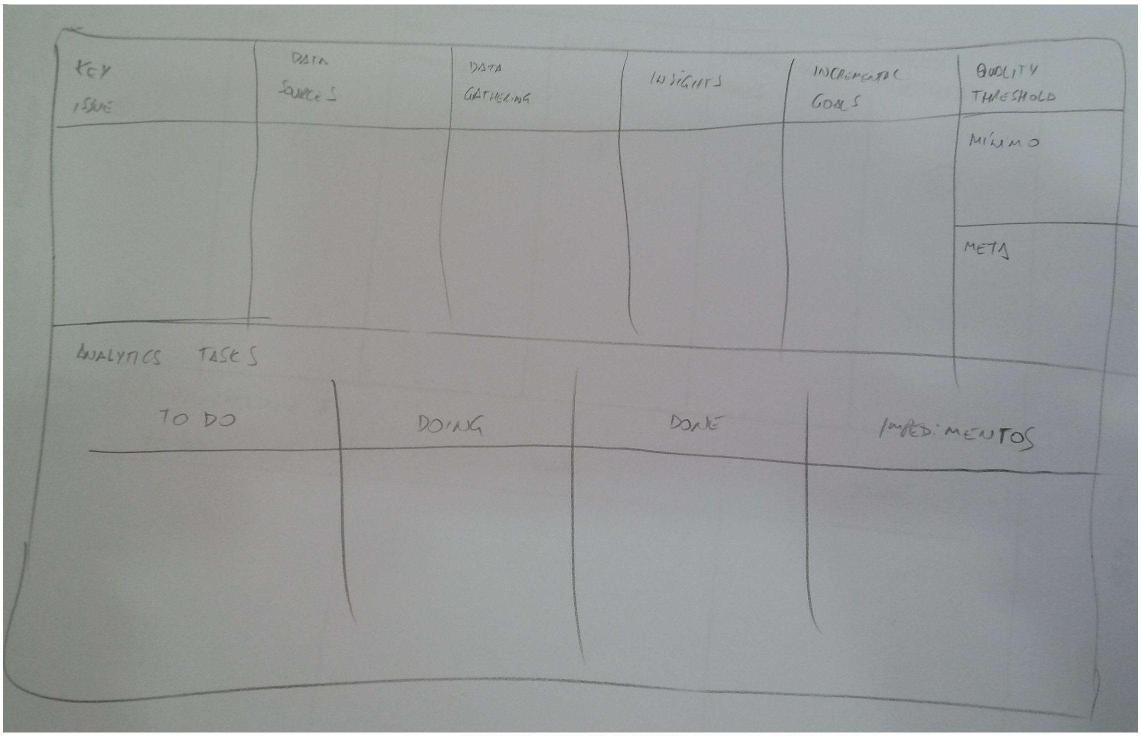
7.1 Participatory design findings

In the first step of the DP, participants were asked about the components of the canvas. The responses of the participants for each SA canvas component are presented in Appendix F. When summarizing the participants' suggestions, we found the need to better clarify at least three canvas components:

- In "Insights", three participants suggested making it clear that insights should be described from the results of the analysis.
- The name of the "Quality Thresholds" block seems not to be suitable. Participants suggested making it clear that the values may be minimum or maximum.
- Understanding what should be considered in "Incremental Goals" was difficult for them. Some have suggested a more appropriate name since the main idea is to implement the improvements.

Also at first step, the participants were asked to individually provide a sketch as a proposal for redesigning the screen. In total, six sketches were generated at the end of this stage. In the second step, they consolidated their opinions with their peers by drawing a new sketch for canvas, generating a sketch per pair. Lastly, in the third step, all participants discussed their redesign proposals and designed a single sketch to the canvas. The final proposal sketched by participants is presented in Figure 7.1.

Figure 7.1 - Final sketch produced during the participatory design session.



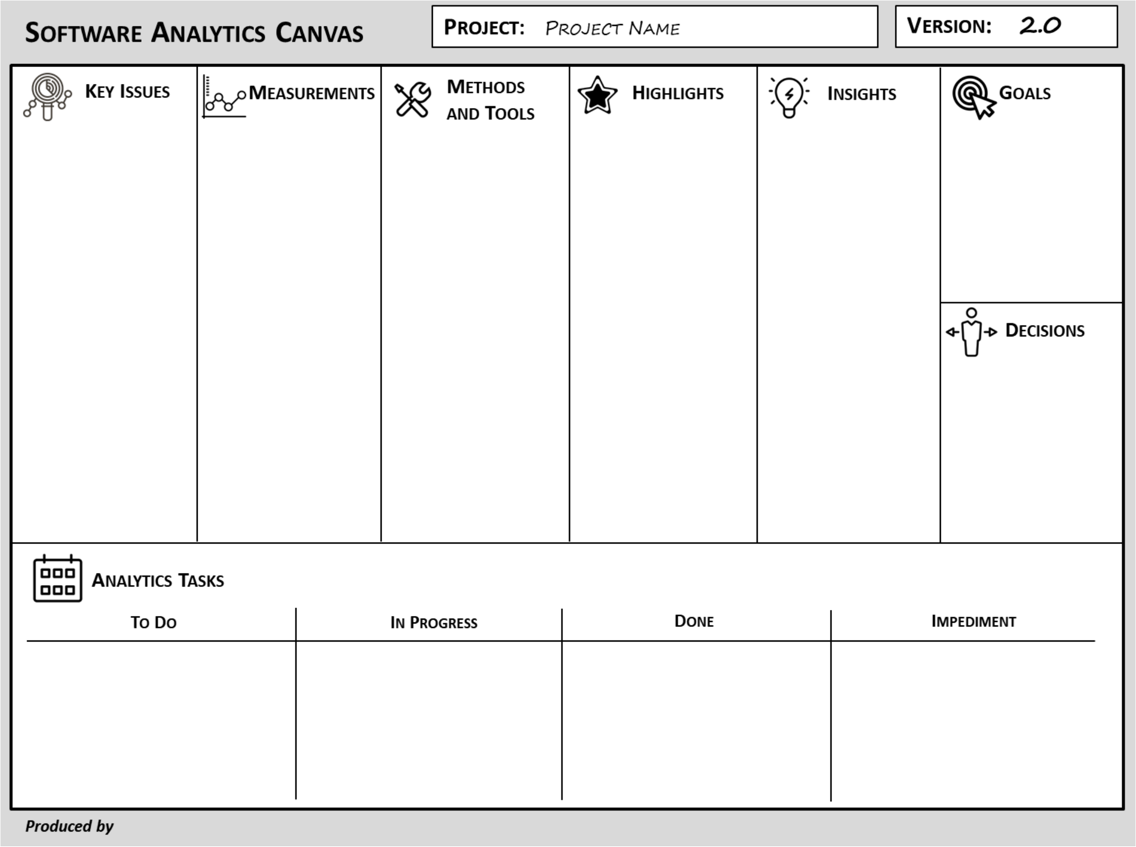
SOURCE: Prepared by study participants.

As Figure 7.1 shows, at the top of the sketch, the participants suggested five blocks reserved for planning (inputs and outputs) named “Key Issues”, “Data Sources”, “Data Gathering”, “Incremental Goals”, and “Quality Threshold”. The latter was subdivided into two parts to include a minimum acceptable value (minimum) and the goal to achieve (goal). At the bottom of the canvas, the participants suggested a block named “Analitycs Tasks” block instead of “Analytics Implementation”, which was subdivided into four parts to accommodate the tasks to do, in progress, done, and the impediments.

7.2 SA Canvas upgrade

We redesigned the SA Canvas considering the findings obtained from observational study as well as the participants’ opinion and suggestions. The second version of the canvas was very similar to the participants’ suggestion sketched during the participatory design session (see Figure 7.1). Figure 7.2 presents the new version of canvas. The new template has 7 components at the top, and 4 components at the bottom.

Figure 7.2 - SA Canvas [version 2.0].



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

SOURCE: Prepared by the author.

As suggested by the participants, we decided to keep the components related to the planning of activities at the top and the components related to the progress of the analytics tasks as an implementation roadmap at the bottom. At the top, *Key Issues* was the only component that have been kept with their original characteristics and name. Regarding other components, we decided to rename *Data Source* to *Measurements* and *Data Gathering* to *Methods and tools* in order to reduce the semantic distance between the names and their definitions. Measurements refer to what will you measure regardless of how it will be measured, while *Methods and tools* refer to means used for conducting the measurements.

Following the suggestions of the participants, when trying to improve the description of the *Insights* component, we realized that there was a step before the identification of the insights that was not being addressed in the previous version, which we named *Highlights*.

Highlights refer to the important observations and information that call attention in the first contact with the data. That is, the collected data have not yet been analyzed in-depth, but at first sight, some findings are noteworthy. In an issue involving code quality and collecting coupling metrics, for example, one can observe that certain classes have a higher coupling than others. In this case, if the team deems this information important, it could be noted as a highlight. In another example involving analysis of architectural metrics, such as the execution time, someone can verify an unstable execution at a certain time of the day, or a determined method presenting a much higher number of calls than others, these can be considered possible highlights.

From the definition of the *Highlights*, the *Insights* component gains a new description. In the new description, insights are identified by analyzing the cause of the points that were highlighted by the team. Considering the previous examples, the team may conclude that coupling is high in a certain part of the code because of business rules that should not be there. And in the case of performance, they conclude that a method is being called unnecessarily, or that it involves a component that depends on the hardware.

The *Incremental Goals* component that had the function of tracking the implementation of achievable goals, and the *Quality Thresholds* component that had the function of setting goals were replaced by other components. In our observational study, we found that participants had difficulty understanding both components. In their place, we created two new components: *Goals* and *Decisions*.

Goals are related to monitoring and control actions, and usually related to some metric. In Goals, values are established to be achieved based on predefined quality indicators. In the coupling example, a goal could be to establish a limit for coupling metrics, and in the performance example it would be to establish a minimum execution time for certain methods. *Decisions* include the actions that will be taken to resolve or minimize a given problem after understanding its origins and reflecting on possible solutions. In the coupling example, a team decision could be to refactor the most problematic part of the code. In the performance example, a decision would be to exchange a certain component of the application.

Regarding the components related to the progress of the analytics tasks, we divided the bottom of the canvas into four blocks to include tasks to be done, tasks in progress, tasks completed, and impediments. It is important to highlight that, these blocks are optional, if the team prefers to keep a single tool to manage the analytics tasks together with development tasks, within the same backlog.

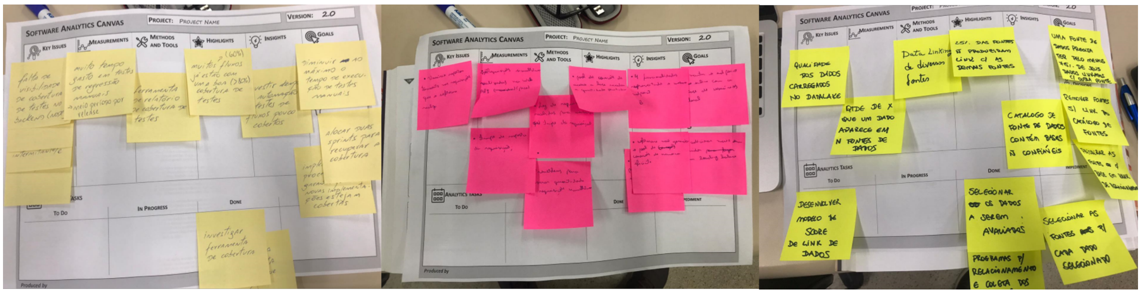
7.3 Second round of evaluation

This section describes the efforts made to evaluate the second version of the canvas. After completing the project for the second version, we looked for partners in the industry to evaluate the use of canvas in practice. In March 2019, we invited 40 software practitioners from INPE and other companies to attend a lecture with the aim of introducing them to SA Canvas and engaging them to participate in a case study. Of the guests, 34 attended the lecture: 16 participants from INPE (TerraME (1) FIP/Terrabrasilis (3), CPTEC (4), EMBRACE (5), and CAP (3)); and 18 participants from other companies (Aeronautics Computer Center-CCA (13), Geopixel (2), ICEA (1), Embraer (1), and Superclient (1)).

7.3.1 Workshops on SA Canvas

Another initiative to evaluate the canvas was to collect data from two workshops. The workshops were held with software industry professionals, in January and February 2020. During the workshops, after the presentation of the *software analytics* approach, the participants were divided into groups of 3-4 people. Using a paper SA Canvas template and stick notes, each group had 10-20 min to raise some key issues based in their work experiences. Then, they had 30-40 min to choose one of the key issues and complete the remaining blocks of the canvas. Finally, each group had 15-20 min to present their results. Figure 7.3 shows some of the canvas templates that were filled out by participants.

Figure 7.3 - SA Canvas filled out by the workshop participants.



SOURCE: Prepared by the author.

After workshops, we surveyed the participants' opinion about the use of SA Canvas through a questionnaire. The questionnaire consisted of both open-ended and closed-ended questions divided in four groups of questions in addition to the demographic questions about the participants: (a) 4 closed-ended questions about the participant's perception of the group's performance when using the canvas, (c) 3 closed-ended questions about the participants' understanding about canvas elements, (d) 3 closed-ended questions about the usability of the canvas, and (e) 3 open-ended questions regarding participants opinion the canvas. For closed-ended questions, we used a 1-5 Likert scale. From two workshops, we collected 15 responses.

The Table 7.1 shows the characterization of the participants in terms of background and professional experience. About 60% of the participants had more than 4 years of experience in software engineering, and more than half of them had intermediate experience in agile practices.

Table 7.1 - Group performance using SA Canvas.

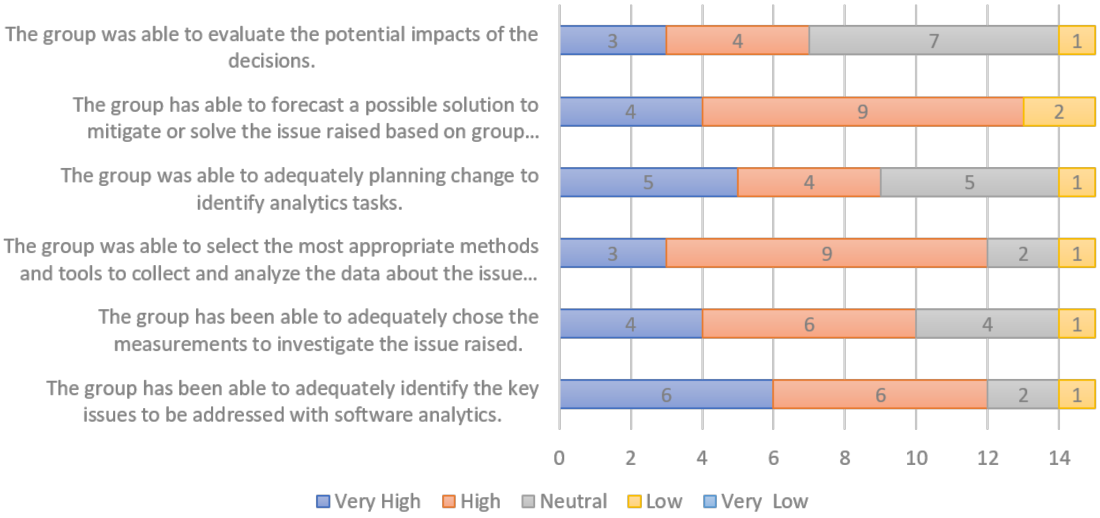
ID	**Background**	**Software Development***	**Agile Practices**
P1	Software Engineer	6 or more	Intermediate
P2	Agilist	6 or more	Expert
P3	Agile Master	0 to 1	Novice
P4	Software Engineer	2 to 4	Intermediate
P5	Chief Technical Officer	4 to 6	Intermediate
P6	Software Engineer	6 or more	Intermediate
P7	Data Specialist	6 or more	Intermediate
P8	Analyst Developer	0 to 1	Novice
P9	Scrum Master	6 or more	Expert
P10	Back-end Developer	6 or more	Intermediate
P11	Software Engineer	1 to 2	Novice
P12	Software Developer	4 to 6	Intermediate
P13	Software Developer	4 to 6	Novice
P14	Software Developer	2 to 4	Novice
P15	Mathematical Modeling Analyst	2 to 4	Intermediate

* Experience time in years

SOURCE: Prepared by the author.

Figure 7.4 presents the bar graph with the participants responses about their perception on group performance by using the SA Canvas.

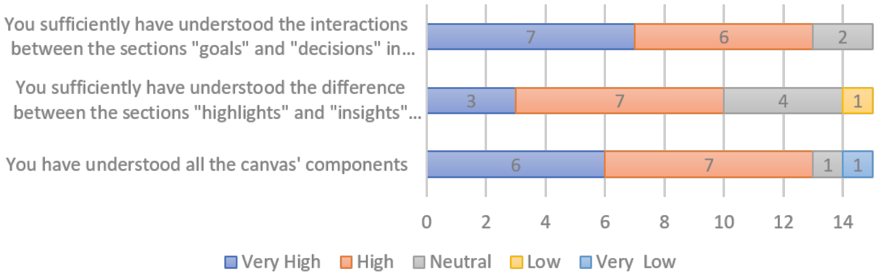
Figure 7.4 - Participants' perception of canvas usage.



SOURCE: Prepared by the author.

As shown in Figure 7.4, for most respondents the groups succeeded to adequately identify the key issues, chose the measurements, select the most appropriate methods and tools, and forecast a possible solution to mitigate or solve the issue raised based. However, a considerable number of participants (above 30%) were unable to assess whether the groups were able to adequately plan the analytical tasks or assess the potential impacts of the decisions. This result is understandable, due to time constraints and the fact that they are not within a real environment. Figure 7.5 presents the bar graph about the participants' understanding of canvas components.

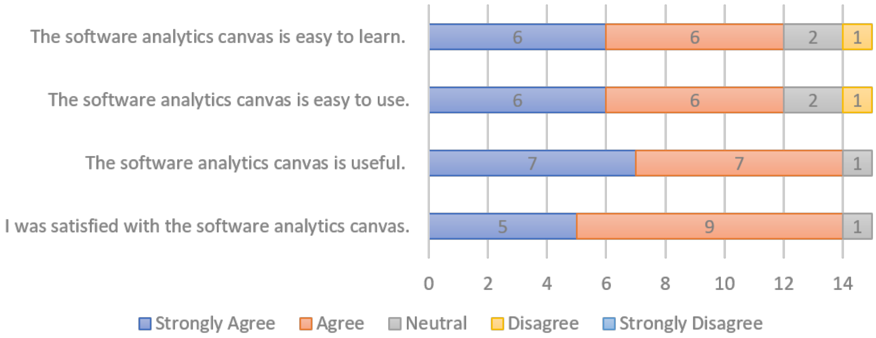
Figure 7.5 - Understanding of canvas components.



SOURCE: Prepared by the author.

Overall, respondents understood all the components and the difference between "Goals" and "Decisions". However, we can note that perhaps the most problematic issue for some of them was to understand the difference between "Highlights" and "Insights". Regarding participants' perception of SA Canvas usability, the most of respondents agreed that the canvas is useful, easy to use, and easy to learn, as shown in Figure 7.6.

Figure 7.6 - Participants' perception of canvas usability.



SOURCE: Prepared by the author.

We asked the participants if they thought the components of the canvas made sense to them, or if any needed to be improved. Some participants pointed out the need to better clarify the components *Highlights* (P4), *Insights* (P10 and P11), *Goals* (P10 and P15), and *Decision* (P3 and P15). P9 suggested merging *Measurements* with *Methods and Tools*. In the P11's opinion, Measurement is "the solution to be implemented". P15 suggested dividing the canvas into two parts, one for defining the problem and action plan, and another for reviewing the plan. Also, we asked the participants what is the main benefit of the canvas for projects related to *software analytics*, in their viewpoint. The responses of the participants are summarized below:

- visibility into problems, activities and goals (P1, P6, P7, P10), allowing for quicker responses (P1)
- organization of ideas (P2, P14) and work steps (P5, P11, P10)
- decision making based on metrics (P4, P12, P13) and fact analysis (P9)
- focus on problem solving (P1, P5, P8, P15)

- focus on action planning (P2, P15) and goals to improve (P3, P8, P13)

Finally, we asked the participants which factors could hinder the adoption of the canvas in practice. One-third of the participants replied that they would have no problem applying the canvas in practice. The other participants pointed out some potential challenges:

- Cultural aspects (P2, P5, P10)
- Lack of team engagement (P10, P15) and support from managers (P12)
- Demand for time to elaborate (P8) and obtain results (P7)
- Lack of knowledge about *software analytics* (P5, P13)
- Lack of interest in management tools (P12, P15)
- Difficulties in fitting into the process (P5)
- Lack of team experience (P15)
- Project profile (P3)

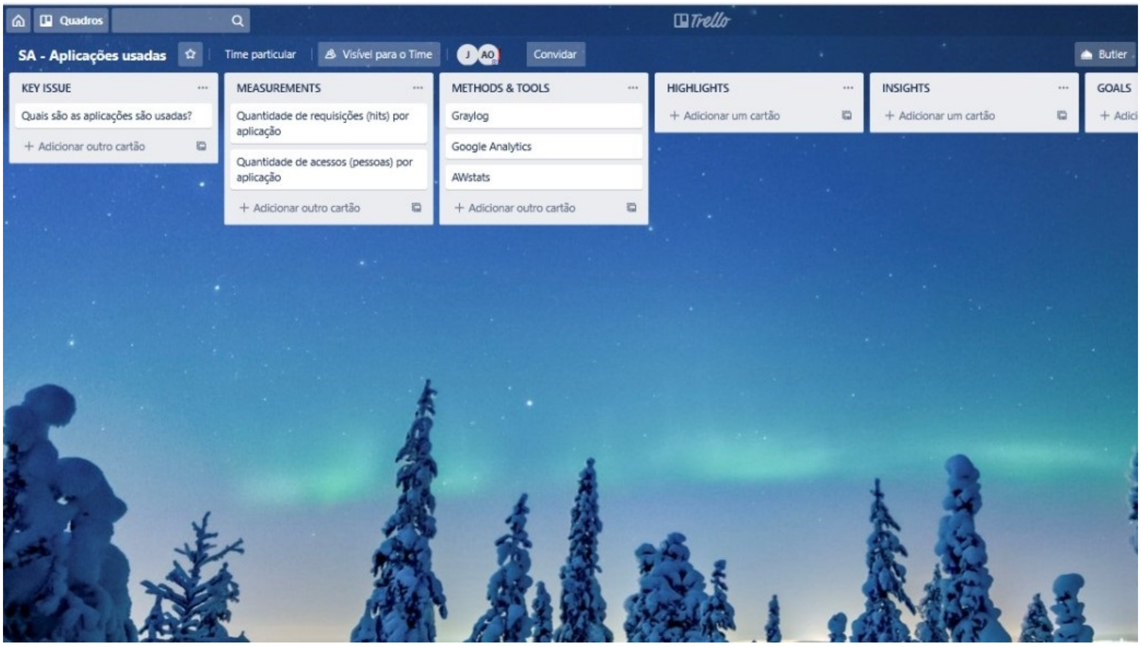
Overall, the feedback received from software professionals regarding the second version reveals the potential of our approach to supporting *software analytics* activities. It may be that some elements still need to be reevaluated or the canvas needs to be adapted to different situations according to the characteristics of the software projects and teams. However, further studies are needed to verify the applicability in long-term projects with different characteristics.

7.3.2 Application at the CPTEC

At the end of 2019, the team from CPTEC at INPE made themselves available to participate in our study. On December 26th, we started a study with the CPTEC team in Cachoeira Paulista.

In the first face-to-face meeting, we presented the canvas to the team and collected some information about their processes. At the end of the presentation, the seven-team members started a discussion about what types of problems could be addressed using Analytics. The next day, we scheduled an online meeting where the team prepared the canvas template in Trello, as shown in Figure 7.7.

Figure 7.7 - SA Canvas organized in Trello.



SA - Aplicações usadas

KEY ISSUE
- Quais são as aplicações são usadas?

MEASUREMENTS
- Quantidade de requisições (hits) por aplicação
- Quantidade de acessos (pessoas) por aplicação

METHODS & TOOLS
- Graylog
- Google Analytics
- AWstats

HIGHLIGHTS

INSIGHTS

GOALS

SOURCE: Prepared by the author.

First, they identified some key issues to be worked on in the following months. Some issues raised by them were: What is the profile of our users? What are the bugs existing in the applications' features? What content is most accessed by users? What applications are being used/accessed most often? How much do our applications consume computational resources?

Thereafter, the team decided to focus on the question about the most used applications. In the sequence, they defined what would be measured (i.e., quantities of requests and accesses), and chose some tools (i.e, Google analytics, Graylog, and AWStats).

As partial results from interactions with them, we figure out that the approach should not be limited to a physical artifact. It is necessary to make it clear that it can be adapted taking into account the needs and resources of the team. For example, the team opted by adapting of the items on the canvas within the same online tool used for planning their development tasks. However, the original canvas template was very important to convey the concepts of the *software analytics*, introduce the canvas components, and explain the information flow.

Moreover, we found that the proposed approach can be used both as a planning tool to track the activities that make up the *software analytics* process and as a research framework to support researchers in the SA area.

When we asked the team leader about his opinion on the components of the canvas, we found that all components made sense for him and none needed to be improved. When we asked about the main benefit of the canvas, the team leader pointed out the way to structure the tasks giving transparency to the team about the process.

As for the impediments or difficulties in adopting the SA Canvas in practice, he pointed out that *"the impediments are not related to Canvas, but to software analytics. We have to reorganize the routine to be able to insert the tasks of SA, this is not simple to do when there is a great demand for work. But it is the burden to achieve the bonus coming from the SA"*.

Their work was in progress, but it was unfortunately interrupted from March onward, due to the Covid-19 pandemic and other demands. Just like the study at CPTEC, other case studies that we were planning to carry out with the EMBRACE and CCST teams were also interrupted for the same reason.

7.4 Chapter summary

This chapter introduces the second version of the canvas explaining the changes made and the new components. Also, it describes the initiatives undertaken to evaluate this new version, such as the study case with the CPTEC team and the workshops carried out with software practitioners. Through these initiatives, we were able to collect the perceived benefits of the approach and the possible difficulties of applying it in practice. Although our approach's evaluation has been positive, we found that some points still need to be checked in future studies.

8 CONCLUDING REMARKS AND FUTURE WORK

This chapter concludes this thesis by reviewing the research questions and introducing the main remarks about the existing research gaps and the open questions in the *software analytics* area presented in Chapter 2, the *software analytics* pattern language presented in Chapter 4, and the SA Canvas artifact presented in the Chapters 5, 6 and 7. In addition, this chapter point out the main contributions this thesis and the suggestions for future work.

This chapter is structured as follows: Section 8.1 contains a brief review of our research questions and findings. Section 8.2 highlights the main contributions and 8.3 addresses directions for future work.

8.1 Summary of the findings

This thesis aimed to propose an approach to supporting agile teams in *software analytics* activities. In the context of INPE projects that have small software teams and that normally adopt agile practices, such an approach would be especially useful to help them make informed decisions in research-related projects, which have a character of exploitation of the domain and the results.

To achieve the thesis objectives, we first conducted a systematic mapping in the existing studies to discover research gaps and the kind of issues commonly addressed by researchers and software professionals in the *software analytics* area (RQ1). We found that there are few studies related to the implementation of *software analytics* projects in practice, especially involving small teams.

From some experience reports, we identified good practices could assist software professionals in conducting *software analytics* projects (RQ2). The patterns language for *software analytics* raised as a proposal to guide agile teams in the planning and conducting of *software analytics* activities in a lighter way.

When presenting the SA patterns to the EMBRACE team, we found that professionals were interested in introducing *software analytics* in their development roadmap. However, we have identified a need to systematize and make the application of these patterns more practical and traceable. Then, our proposal was to consolidate the patterns for *software analytics* within a structure in canvas format to support professionals activities (RQ3).

The first version of SA Canvas was designed and evaluated in a laboratory setting. The study objectives were to evaluate the artifact in use through an observational study, collected the participants' opinions on usefulness and ease of use using a questionnaire, and held a participatory design session to collect suggestions for improvement in the design of the artifact.

The observational study was specifically carried out to understand how the SA Canvas could support cognitive activities during the planning and management of *software analytics* activities. To analyze the observational data, we identified what were the actions that characterized each of the participants' strategies throughout the three iterations using an approach based on distributed cognition and sequential analysis.

In our analysis, we first categorized the interaction events into emergent actions, and then we associated these actions with the interaction strategies of the Resource Model (WRIGHT et al., 2000). Then, we analyzed the sequencing of interactions in the temporal view to map the interaction strategy, components of the canvas, and other artifacts. Finally, we performed a sequential analysis (SANDERSON; FISHER, 1994) to identify common patterns on the shift between interaction strategies that occurred during each iteration. We found two main patterns through sequential analysis carried out to discover strategy exchange patterns during participants' interactions with available resources (see Figure 6.9 in Chapter 6).

Outside the findings of the cognitive aspects analysis, we verified that the canvas worked well as a hub in terms of the information flow involving *software analytics* issues. Information hubs can be considered spaces where information flows meet and decisions are made. The proposed artifact is also a situation awareness channel, which considers how people are kept informed about what is happening (BERNDT et al., 2015). For Agile teams, the support of different visualization techniques and tools throughout the software development process is crucial to foster awareness and communication. Additionally, this factor can have a positive impact on the sense of purpose (LIECHTI et al., 2017b) when the professionals follow the evolution of their actions within a cycle of continuous improvement.

Despite the improvements that are needed, we found that the canvas design meets the principle of *representation* that considers one way in which external artifacts can aid cognition is by providing an explicit representation of the relationship between the current state and a goal state (WRIGHT et al., 2000).

The closer the representation is to the cognitive need or goal of the user, the more powerful that representation will be. In the case of SA Canvas, the team's goal is often to keep the focus on solve issues previously defined. Over time, results are achieved and new goals are defined.

Our method of analysis was very useful at exposing the cognitive aspects of participants' activities. When we analyzed the actions and sequencing of interactions over time, we were able to identify, for instance, that groups had difficulties in understanding some components of the canvas when sought for the document with canvas guidelines. Then, after a brief discussion or shared ideas, they were able to complete their tasks. Many difficulties observed were later confirmed with the collection of opinion from the participants and the results of the participatory design, as presented in Chapter 6. Reflecting on the findings of this first evaluation cycle, we redesigned and proposed the second canvas version.

We started a new evaluation round of the second version and obtained some results through case studies and workshops. We had planned to conduct more studies with the teams at INPE. However, with the suspension of activities due to the covid-19 pandemic, we would not have time to redirect our studies within the research period. From obtained results, we found that our approach is promising, but we recognized that it needs to be evaluated in more long-term projects with different characteristics.

8.2 Contributions

The main contribution of this thesis is SA Canvas proposed as an approach to supporting agile teams planning and managing *software analytics* activities. SA Canvas can be considered a goal-focused approach, similar to the well-known Goal Question Metric (GQM) approach proposed by Basili et al. (BASILI et al., 1994). The goal-focused approaches are good ways to ensure that measurement goals are articulated with the metrics being collected, and also, to avoid having useless measurements. We argue that the SA Canvas is a suitable artifact to agile teams' informative workspaces where various techniques and tools for software visualization are commonly applied as information radiators – e.g., count of velocity automated tests, continuous integration status, incident reports, and so forth (BECK; ANDRES, 2004).

Other important contributions of this thesis are:

- The literature review presenting the state of the art in the area of *software analytics*, highlighting the main issues addressed in the area, and pointing out the main research gaps.
- The *software analytics* patterns proposed as a way to encourage small and medium-sized Agile teams to incorporate data-based approaches into their development process to make better decisions.
- The methodology of analysis based on the Resource Model (WRIGHT et al., 2000) and the sequential analysis (SANDERSON; FISHER, 1994) used to evaluate cognitive activities observed from the study participants' interaction with the canvas.

8.3 Future work

This work open up several opportunities for research into *software analytics* area, such as questions that remain open or little studied. Considering the current stage of our work presented here, some of the future directions consists of conducting studies within the software industry on real projects, involving different segments and software products. At INPE, our approach can be adopted by several development teams with great potential to contribute to quality management and productivity in software development in the space research areas.

Currently, we are following two master's studies in progress at the University of Porto that are using SA Canvas. The former is a case study in a software startup involving multiple teams using our canvas to address the code quality problems in their projects. The study goal is to evaluate the impact the approach has on the teams involved in terms of understanding the code quality issues and metrics implemented but also on the efficiency and effectiveness of the approach. Also in the area of software quality, the other study is being conducted in a larger company that has several software teams. This study aims to investigate the utility of the canvas and also the scalability of the approach in different types of software projects.

Furthermore, recently, we found out that our work has inspired software professionals from other countries. The software engineer specialized in the analysis of software data from Germany, Markus Harrer created a version of the Sofware Analytics Canvas with a smaller number of components. More details of Harrer's canvas is available in (HARRER, 2020).

Also recently, we received an email from a professional who is part of the analytics team at OLX-Br congratulating us on the research, and showing interest in the second version of the canvas. We forwarded the new version of the canvas and made ourselves available for further clarification.

For future work, we would recommend conducting longitudinal studies involving software teams in actual projects to further verify the results of this thesis. Moreover, the use of canvas can be verified in other contexts, such as Lean Analytics in software startups. Other future work would be the identification of new patterns based on the experiences of using SA Canvas, and the development of a tool based on the canvas to support *software analytics* activities remotely. Finally, the methodology using a resource model and sequential analysis for evaluating the canvas in the laboratory could be used in other studies involving cognitive activities and the interaction with software artifacts.

REFERENCES

AALST, W. v. d. Big software on the run: in vivo software analytics based on process mining. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESS, 2015. **Proceedings...** [S.l.]: ACM, 2015. p. 1–5. 3

ABDELLATIF, T. M.; CAPRETZ, L. F.; HO, D. Software analytics to software practice: a systematic literature review. In: INTERNATIONAL WORKSHOP ON BIG DATA SOFTWARE ENGINEERING, 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 30–36. 3

AGRAWAL, A.; MENZIES, T.; MINKU, L. L.; WAGNER, M.; YU, Z. Better software analytics via" duo": data mining algorithms using/used-by optimizers. **arXiv preprint arXiv:1812.01550**, 2018. 29, 161, 163

AKTINSON, P.; HAMMERSLEY, M. Ethnography and participant observation. In: DENZIN, N. K.; LINCOLN, Y. S. (Ed.). **Strategies of Qualitative Inquiry**. [S.l.]: Thousand Oaks: Sage, 1998. p. 248–261. 46

ALELYANI, T.; YANG, Y. Software crowdsourcing reliability: an empirical study on developers behavior. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 2., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 36–42. 27, 159, 163

ALEXANDER, C. **A pattern language: towns, buildings, construction**. New York: Oxford University Press, 1977. 54, 60

______. **The timeless way of building**. New York: Oxford University Press, 1979. 53

AN, L.; CASTELLUCCIO, M.; KHOMH, F. An empirical study of dll injection bugs in the firefox ecosystem. **Empirical Software Engineering**, p. 1–24, 2019. 31, 162, 164

AN, L.; KHOMH, F.; GUÉHÉNEUC, Y.-G. An empirical study of crash-inducing commits in mozilla firefox. **Software Quality Journal**, v. 26, n. 2, p. 553–584, 2018. 29, 160, 163

ANICHE, M. F.; OLIVA, G. A.; GEROSA, M. A. Why statically estimate code coverage is so hard? a report of lessons learned. In: BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 29., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 185–190. 31, 158, 164

ANWAR, H.; PFAHL, D. Towards greener software engineering using software analytics: a systematic mapping. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 43., 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 157–166. 3

ARNDT, T. Big data and software engineering: prospects for mutual enrichment. **Iran Journal of Computer Science**, v. 1, n. 1, p. 3–10, 2018. 35, 161, 165

ASUNCION, H. U.; SHONLE, M.; PORTER, R.; POTTS, K.; DUNCAN, N.; MATTHIES, W. Using change entries to collect software project information. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 2013. **Proceedings...** [S.l.], 2013. 27, 157, 163

AUGUSTINE, V.; HUDEPOHL, J.; MARCINCZAK, P.; SNIPES, W. Deploying software team analytics in a multinational organization. **IEEE Software**, v. 35, n. 1, p. 72–76, 2018. 39, 42, 161, 165

BAGHERZADEH, M.; KAHANI, N.; BEZEMER, C.-P.; HASSAN, A. E.; DINGEL, J.; CORDY, J. R. Analyzing a decade of linux system calls. **Empirical Software Engineering**, v. 23, n. 3, p. 1519–1551, 2018. 25, 160, 163

BAKEMAN, R.; GOTTMAN, J. M. **Observing interaction: an introduction to sequential analysis**. [S.l.]: Cambridge University Press, 1997. 50, 100, 101, 107, 111

BAKER, L. Observation: a complex research method. **Library Trends**, v. 55, n. 1, p. 171–189, 2006. 46

BALDASSARI, B. Squore: a new approach to software project quality measurement. In: INTERNATIONAL CONFERENCE ON SOFTWARE & SYSTEMS ENGINEERING AND THEIR APPLICATIONS, 2012. **Proceedings...** [S.l.], 2012. 34, 157, 164

BALTES, S.; KNACK, J.; ANASTASIOU, D.; TYMANN, R.; DIEHL, S. No influence of continuous integration on the commit activity in github projects. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 4., 2018. **Proceedings...** [S.l.], 2018. p. 1–7. 26, 160, 163

BAO, L.; XING, Z.; XIA, X.; LO, D.; HASSAN, A. E. Inference of development activities from interaction with uninstrumented applications. **Empirical Software Engineering**, v. 23, n. 3, p. 1313–1351, 2018. 32, 161, 164

BARBOUR, L.; AN, L.; KHOMH, F.; ZOU, Y.; WANG, S. An investigation of the fault-proneness of clone evolutionary patterns. **Software Quality Journal**, v. 26, n. 4, p. 1187–1222, 2018. 29, 160, 163

BARIK, T.; DELINE, R.; DRUCKER, S.; FISHER, D. The bones of the system: a case study of logging and telemetry at microsoft. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING COMPANION, 38., 2016. **Proceedings...** [S.l.]: IEEE/ACM, 2016. p. 92–101. 33, 159, 164

BASILI, V. R.; CALDIERA, G.; ROMBACH, D. H. The goal question metrics approach. In: MARCINIAK, J. J. (Ed.). **Encyclopedia of Software Engineering**. [S.l.]: John Wiley & Sons, 1994. v. 1, p. 528–532. 126

BASKERVILLE, R. **What design science is not**. [S.l.]: Taylor & Francis, 2008. 43

BASTIAN, M.; HEYMANN, S.; JACOMY, M. Gephi: an open source software for exploring and manipulating networks. In: INTERNATIONAL CONFERENCE ON WEBLOGS AND SOCIAL MEDIA, 3., 2009. **Proceedings...** [S.l.]: AAAI, 2009. 21

BATARSEH, F. A.; GONZALEZ, A. J. Predicting failures in agile software development through data analytics. **Software Quality Journal**, v. 26, n. 1, p. 49–66, 2018. 25, 161, 163

BAYATI, S.; PARSONS, D.; SUSNJAK, T.; HEIDARY, M. Big data analytics on large-scale socio-technical software engineering archives. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY, 3., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 65–69. 31, 158, 164

BAYSAL, O. Informing development decisions: from data to information. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 1407–1410. 36, 87, 157, 164

BAYSAL, O.; HOLMES, R.; GODFREY, M. W. Developer dashboards: the need for qualitative analytics. **IEEE Software**, v. 30, n. 4, p. 46–52, 2013. 3, 36, 86, 157, 165

______. Situational awareness: personalizing issue tracking systems. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 1185–1188. 69

BAYSAL, O.; KONONENKO, O.; HOLMES, R.; GODFREY, M. W. Extracting artifact lifecycle models from metadata history. In: INTERNATIONAL WORKSHOP ON DATA ANALYSIS PATTERNS IN SOFTWARE ENGINEERING, 1., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 17–19. 56

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change**. [S.l.]: Addison-Wesley Professional, 2004. 85, 126

BECK, K. et al. Manifesto for agile software development. 2001. Disponível em: <`https://agilemanifesto.org/`>. 4, 84

BEGEL, A.; ZIMMERMANN, T. Analyze this! 145 questions for data scientists in software engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 36., 2014. **Proceedings...** [S.l.]: ACM, 2014. p. 12–23. 35, 158, 164

BERNDT, E.; FURNISS, D.; BLANDFORD, A. Learning contextual inquiry and distributed cognition: a case study on technology use in anaesthesia. **Cognition, Technology & Work**, v. 17, n. 3, p. 431–449, 2015. 125

BODKER, S.; GRONBÆK, K.; KYNG, M. Cooperative design: techniques and experiences from the scandinavian scene. In: GRUDIN, J. (Ed.). **Readings in Human–Computer Interaction**. [S.l.]: Elsevier, 1995. p. 215–224. 47

BRATTETEIG, T.; BøDKER, K.; DITTRICH, Y.; MOGENSEN, P. H.; SIMONSEN, J. **Methods: organising principles and general guidelines for Participatory Design projects.** [S.l.]: Routledge, 2012. 117-144 p. 47

BROWN, J. M.; LINDGAARD, G.; BIDDLE, R. Collaborative events and shared artefacts: agile interaction designers and developers working toward common aims. In: AGILE CONFERENCE, 2011. **Proceedings...** [S.l.]: IEEE, 2011. p. 87–96. 85

BRUNTINK, M. An initial quality analysis of the ohloh software evolution data. In: INTERNATIONAL WORKSHOP ON SOFTWARE QUALITY AND MAINTAINABILITY, 2014. **Proceedings...** [S.l.]: Citeseer, 2014. v. 65. 24, 157, 163

______. Towards base rates in software analytics: early results and challenges from studying ohloh. **Science of Computer Programming**, v. 97, p. 135–142, 2015. 34, 158, 164

BUSCHMANN, F.; HENNEY, K.; SCHIMDT, D. **Pattern-oriented software architecture: on patterns and pattern language**. [S.l.]: John Wiley & Sons, 2007. 54, 56

BUSE, R. P.; ZIMMERMANN, T. Analytics for software development. In: WORKSHOP ON FUTURE OF SOFTWARE ENGINEERING RESEARCH, 2010. **Proceedings...** [S.l.]: FSE/SDP, 2010. p. 77–80. 1, 3, 4, 10, 11, 16

______. Information needs for software development analytics. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 34., 2012. **Proceedings...** [S.l.], 2012. p. 987–996. 10, 11, 16, 56

CATAL, C. Software mining and fault prediction. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, v. 2, n. 5, p. 420–426, 2012. 29, 157, 163

CERULO, L.; PENTA, M. D.; BACCHELLI, A.; CECCARELLI, M.; CANFORA, G. Irish: a hidden markov model to detect coded information islands in free text. **Science of Computer Programming**, v. 105, p. 26–43, 2015. 3, 32, 64, 158, 164

CHAFFEY, D.; PATRON, M. From web analytics to digital marketing optimization: increasing the commercial value of digital analytics. **Journal of Direct, Data and Digital Marketing Practice**, v. 14, n. 1, p. 30–45, 2012. 1

CHATZIKONSTANTINOU, G.; KONTOGIANNIS, K.; ATTARIAN, I.-M. A goal driven framework for software project data analytics. In: INTERNATIONAL CONFERENCE ON ADVANCED INFORMATION SYSTEMS ENGINEERING, 2013. **Proceedings...** [S.l.]: Springer, 2013. p. 546–561. 27, 157, 163

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; GHOSE, A. Predicting delays in software projects using networked classification (t). In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 30., 2015. **Proceedings...** [S.l.]: IEEE/ACM, 2015. p. 353–364. 30, 159, 163

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; PHAM, T. T. M.; GHOSE, A.; MENZIES, T. A deep learning model for estimating story points. **IEEE Transactions on Software Engineering**, 2018. 29, 160, 163

CHOMA, J.; GUERRA, E. M.; SILVA, T. S. Patterns for implementing software analytics in development teams. In: CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 24., 2017. **Proceedings...** [S.l.], 2017. p. 12. 53, 56, 97

_____. Learning from experiments, define quality standards, suspend measurement: three patterns in a software analytics pattern language. In: LATIN AMERICAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 12., 2018. **Proceedings...** [S.l.], 2018. p. 10. 53, 56, 97

CHOMA, J.; GUERRA, E. M.; SILVA, T. S. da; ZAINA, L. A.; CORREIA, F. F. Towards an artifact to support agile teams in software analytics activities. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 2019. **Proceedings...** [S.l.], 2019. 88

CITO, J. Developer targeted analytics: supporting software development decisions with runtime information. In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 31., 2016. **Proceedings...** [S.l.]: IEEE/ACM, 2016. p. 892–895. 34, 159, 164

CITO, J.; OLIVEIRA, F.; LEITNER, P.; NAGPURKAR, P.; GALL, H. C. Context-based analytics: establishing explicit links between runtime traces and source code. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 39., 2017. **Proceedings...** [S.l.], 2017. p. 193–202. 25, 159, 163

COCKBURN, A. **Crystal clear: a human-powered methodology for small teams**. [S.l.]: Pearson Education, 2004. 85

COES, B. **Critically assessing the strengths and limitations of the Business Model Canvas.** 2014. 99 p. Master thesis Business Administration — University of Twente, Nijverdal, 2014. 87

COHN, M. **User stories applied: for agile software development**. [S.l.]: Addison-Wesley Professional, 2004. 85

_____. **Agile estimating and planning**. [S.l.]: Pearson Education, 2005. 85

CONBOY, K.; DENNEHY, D.; O'CONNOR, M. Big time: an examination of temporal complexity and business value in analytics. **Information & Management**, 2018. 36, 161, 165

COPLIEN, J. O.; HARRISON, N. **Organizational patterns of agile software development**. [S.l.]: Pearson Prentice Hall Upper Saddle River, 2005. 54, 75

COPLIEN, J. O.; SCHMIDT, D. C. **Pattern languages of program design**. [S.l.]: ACM Press/Addison-Wesley Publishing, 1995. 54

COSENTINO, V.; DUEñAS, S.; ZEROUALI, A.; ROBLES, G.; GONZALEZ-BARAHONA, J. M. Graal: The quest for source code knowledge. In: INTERNATIONAL WORKING CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION, 18., 2018. **Proceedings...** [S.l.], 2018. p. 123–128. 25, 160, 163

COSENTINO, V.; IZQUIERDO, J. L. C.; CABOT, J. A systematic mapping study of software development with github. **IEEE Access**, v. 5, p. 7173–7192, 2017. 22

CZECH, M.; HÜLLERMEIER, E.; JAKOBS, M.-C.; WEHRHEIM, H. Predicting rankings of software verification competitions. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 3., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 23–26. 28, 160, 163

CZERWONKA, J.; NAGAPPAN, N.; SCHULTE, W.; MURPHY, B. Codemine: building a software development data analytics platform at microsoft. **IEEE Software**, v. 30, n. 4, p. 64–71, 2013. 38, 157, 165

DAM, H. K.; TRAN, T.; GHOSE, A. Explainable software analytics. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 40., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 53–56. 30, 161, 163

DAM, H. K.; TRAN, T.; GRUNDY, J.; GHOSE, A. Deepsoft: a vision for a deep model of software. In: INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 24., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 944–947. 28, 159, 163

DANG, Y.; ZHANG, D.; GE, S.; HUANG, R.; CHU, C.; XIE, T. Transferring code-clone detection and analysis to practice. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 39., 2017. **Proceedings...** [S.l.], 2017. p. 53–62. 35, 160, 165

D'ASTOUS, P.; ROBILLARD, P. N. Empirical study of exchange patterns during software peer review meetings. **Information and Software Technology**, v. 44, n. 11, p. 639–648, 2002. 46, 50

DAVENPORT, T. H. Make better decisions. **Harvard Business Review**, v. 87, n. 11, p. 117–123, 2009. 1

DAVENPORT, T. H.; HARRIS, J. G. **Competing on analytics: the new science of winning**. [S.l.]: Harvard Business Press, 2007. 1

DAVIS, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. **MIS Quarterly**, p. 319–340, 1989. 95, 99

DECAN, A.; MENS, T.; GROSJEAN, P. An empirical comparison of dependency network evolution in seven software packaging ecosystems. **Empirical Software Engineering**, v. 24, n. 1, p. 381–416, 2019. 34, 162, 164

DEHGHAN, A.; BLINCOE, K.; DAMIAN, D. A hybrid model for task completion effort estimation. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 2., 2016. **Proceedings...** [S.l.], 2016. p. 22–28. 29, 159, 163

DEISSENBOECK, F.; JUERGENS, E.; HUMMEL, B.; WAGNER, S.; PARAREDA, B. M. Y.; PIZKA, M. Tool support for continuous quality control. **IEEE Software**, v. 25, n. 5, p. 60–67, 2008. 84, 86

DESHPANDE, A.; SHARP, H.; BARROCA, L.; GREGORY, P. Remote working and collaboration in agile teams. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS, 2016. **Proceedings...** [S.l.], 2016. p. 11–24. 48

DEVANBU, P.; KUDIGRAMA, P.; RUBIO-GONZÁLEZ, C.; VASILESCU, B. Timezone and time-of-day variance in github teams: an empirical method and study. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 3., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 19–22. 27, 160, 163

DUBOCHET, G. Computer code as a medium for human communication: are programming languages improving? In: WORKING CONFERENCE ON THE PSYCHOLOGY OF PROGRAMMERS INTEREST GROUP, 21., 2009. **Proceedings...** [S.l.]: University of Limerick, 2009. p. 174–187. 50

DUEÑAS, S.; COSENTINO, V.; ROBLES, G.; GONZALEZ-BARAHONA, J. M. Perceval: software project data at your will. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 40., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 1–4. 32, 161, 164

ELLMANN, M. On the similarity of software development documentation. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 11., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 1030–1033. 35, 160, 165

ELLMANN, M.; OESER, A.; FUCCI, D.; MAALEJ, W. Find, understand, and extend development screencasts on youtube. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 3., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 1–7. 25, 160, 163

FALESSI, D.; MOEDE, M. J. Facilitating feasibility analysis: the pilot defects prediction dataset maker. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 4., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 15–18. 34, 161, 164

FELDERER, M. Issues on software quality models for mastering change. In: STEFFEN, B. (Ed.). **Transactions on foundations for mastering change I**. [S.l.]: Springer, 2016. p. 225–241. 36, 159, 165

FELDT, R.; STARON, M.; HULT, E.; LILJEGREN, T. Supporting software decision meetings: heatmaps for visualising test and code measurements. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 39., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 62–69. 37, 78, 157, 165

FINLAY, J.; PEARS, R.; CONNOR, A. M. Data stream mining for predicting software build outcomes using source code metrics. **Information and Software Technology**, v. 56, n. 2, p. 183–198, 2014. 32, 158, 164

FLEURY, S.; JAMET, É.; GHORBEL, A.; LEMAITRE, A.; ANQUETIL, E. Application of the resources model to the supervision of an automated process. **Human–Computer Interaction**, v. 30, n. 2, p. 103–121, 2015. 50

FLOR, N. V.; HUTCHINS, E. Analyzing distributed cognition in software teams: a case study of collaborative programming during adaptive software maintenance. In: WORKSHOP ON EMPIRICAL STUDIES OF PROGRAMMERS, 4., 1992. **Proceedings...** [S.l.], 1992. p. 36–64. 48

FOIDL, H.; FELDERER, M. Data science challenges to improve quality assurance of internet of things applications. In: INTERNATIONAL SYMPOSIUM ON LEVERAGING APPLICATIONS OF FORMAL METHODS, 2016. **Proceedings...** [S.l.]: Springer, 2016. p. 707–726. 36, 78, 159, 165

FOTROUSI, F. Quality-impact assessment of software systems. In: INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 24., 2016. **Proceedings...** [S.l.]: IEEE, 2016. p. 427–431. 26, 159, 163

FU, W.; MENZIES, T. Easy over hard: a case study on deep learning. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 11., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 49–60. 32, 159, 164

FURNISS, D.; BLANDFORD, A. Understanding emergency medical dispatch in terms of distributed cognition: a case study. **Ergonomics**, v. 49, n. 12-13, p. 1174–1203, 2006. 47

GAMMA, E. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Pearson Education India, 1995. 53

GERARD, W.; OVERBEEK, S.; BRINKKEMPER, S. Fuzzy artefacts: formality of communication in agile teams. In: INTERNATIONAL CONFERENCE ON THE QUALITY OF INFORMATION AND COMMUNICATIONS TECHNOLOGY), 11., 2018. **Proceedings...** [S.l.]: IEEE, 2018. p. 1–7. 85

GIGER, E.; GALL, H. C. Effect size analysis. In: INTERNATIONAL WORKSHOP ON DATA ANALYSIS PATTERNS IN SOFTWARE ENGINEERING, 1., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 11–13. 55

GODFREY, M. W. Understanding software artifact provenance. **Science of Computer Programming**, v. 97, p. 86–90, 2015. 35, 158, 164

GONZALEZ-BARAHONA, J. M.; IZQUIERDO-CORTAZAR, D.; MAFFULLI, S.; ROBLES, G. Understanding how companies interact with free software communities. **IEEE Software**, v. 30, n. 5, p. 38–45, 2013. 27, 157, 163

GONZÁLEZ-TORRES, A.; GARCÍA-PEÑALVO, F. J.; THERÓN, R. A framework for the evolutionary visual software analytics process. In: LYTRAS, M. D. et al. (Ed.). **Information systems, e-learning, and knowledge management research**. Berlin: Springer, 2013. v. 278, p. 439. 36, 157, 165

______. Human–computer interaction in evolutionary visual software analytics. **Computers in Human Behavior**, v. 29, n. 2, p. 486–495, 2013. 3

GONZÁLEZ-TORRES, A.; GARCÍA-PEÑALVO, F. J.; THERÓN-SÁNCHEZ, R.; COLOMO-PALACIOS, R. Knowledge discovery in software teams by means of evolutionary visual software analytics. **Science of Computer Programming**, v. 121, p. 55–74, 2016. 37, 159, 165

GONZÁLEZ-TORRES, A.; NAVAS-SÚ, J.; HERNÁNDEZ-VÁSQUEZ, M.; SOLANO-CORDERO, J.; HERNÁNDEZ-CASTRO, F. A proposal towards the design of an architecture for evolutionary visual software analytics. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND COMPUTER SCIENCE, 2018. **Proceedings...** [S.l.], 2018. p. 269–276. 37, 160, 165

GONZALEZ-TORRES, A.; THERON, R.; GARCIA-PENALVO, F. J.; WERMELINGER, M.; YU, Y. Maleku: an evolutionary visual software analysis tool for providing insights into software evolution. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 27., 2011. **Proceedings...** [S.l.]: IEEE, 2011. p. 594–597. 67

GOUSIOS, G.; SAFARIC, D.; VISSER, J. Streaming software analytics. In: INTERNATIONAL WORKSHOP ON BIG DATA SOFTWARE ENGINEERING, 2., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 8–11. 24, 75, 159, 163

GREGOR, S.; HEVNER, A. R. Positioning and presenting design science research for maximum impact. **MIS Quarterly**, v. 37, n. 2, 2013. 43

GUERRA, E.; ANICHE, M. Achieving quality on software design through test-driven development. **Software Quality Assurance**, p. 201–220, 2015. 72

GUERRA, E.; SOUZA, J. de; FERNANDES, C. Pattern language for the internal structure of metadata-based frameworks. In: NOBLE, J.; JOHNSON, R.; ZDUN, U.; WALLINGFORD, E. (Ed.). **Transactions on pattern languages of programming III**. [S.l.]: Springer, 2013. p. 55–110. 54, 55

GUO, J.; RAHIMI, M.; CLELAND-HUANG, J.; RASIN, A.; HAYES, J. H.; VIERHAUSER, M. Cold-start software analytics. In: INTERNATIONAL WORKSHOP ON MINING SOFTWARE REPOSITORIES, 13., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 142–153. 3, 27, 159, 163

HARON, N. H.; SYED-MOHAMAD, S. M. Test and defect coverage analytics model for the assessment of software test adequacy. In: SOFTWARE ENGINEERING CONFERENCE, 9., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 13–18. 37, 72, 158, 165

HARRER, M. **Software analytics canvas**. 2020. Disponível em: <`https://www.feststelltaste.de/software-analytics-canvas/`>. 127

HARTMANN, D.; DYMOND, R. Appropriate agile measurement: using metrics and diagnostics to deliver business value. In: AGILE CONFERENCE, 2006. **Proceedings...** [S.l.], 2006. p. 6–pp. 4

HASSAN, A. E.; XIE, T. Software intelligence: the future of mining software engineering data. In: WORKSHOP ON FUTURE OF SOFTWARE ENGINEERING RESEARCH, 2010. **Proceedings...** [S.l.]: ACM, 2010. p. 161–166. 1, 2, 4, 10, 11, 16

HEMMATI, A.; ALAM, S. D. A.; CARLSON, C. Utilizing product usage data for requirements evaluation. In: INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 26., 2018. **Proceedings...** [S.l.]: IEEE, 2018. p. 432–435. 26, 162, 163

HEVNER, A.; CHATTERJEE, S. Design science research in information systems. In: ——(Ed.). **Design research in information systems**. [S.l.]: Springer, 2010. p. 9–22. 44

HEVNER, A. R.; MARCH, S. T.; PARK, J.; RAM, S. Design science in information systems research. **MIS Quarterly**, v. 28, n. 1, p. 75–105, 2004. 43, 44, 45

HINDLE, A.; BIRD, C.; ZIMMERMANN, T.; NAGAPPAN, N. Do topics make sense to managers and developers? **Empirical Software Engineering**, v. 20, n. 2, p. 479–515, 2015. 32, 158, 164

HOLLAN, J.; HUTCHINS, E.; KIRSH, D. Distributed cognition: toward a new foundation for human-computer interaction research. **ACM Transactions on Computer-Human Interaction**, v. 7, p. 174–196, 2000. 47, 48

HUANG, Q.; XIA, X.; LO, D. Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction. **Empirical Software Engineering**, p. 1–40, 2018. 29, 161, 163

HUANG, Y.; COSTA, D. A. da; ZHANG, F.; ZOU, Y. An empirical study on the issue reports with questions raised during the issue resolving process. **Empirical Software Engineering**, p. 1–33, 2018. 30, 160, 163

HUIJGENS, H.; LAMPING, R.; STEVENS, D.; ROTHENGATTER, H.; GOUSIOS, G.; ROMANO, D. Strong agile metrics: mining log data to determine predictive power of software metrics for continuous delivery teams. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 11., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 866–871. 34, 81, 160, 164

HUIJGENS, H.; SPADINI, D.; STEVENS, D.; VISSER, N.; DEURSEN, A. van. Software analytics in continuous delivery: a case study on success factors. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 12., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 25. 39, 42, 161, 165

HULLETT, K.; NAGAPPAN, N.; SCHUH, E.; HOPSON, J. Data analytics for game development. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 33., 2011. **Proceedings...** [S.l.]: IEEE, 2011. p. 940–943. 1

HUTCHINS, E. **Cognition in the wild**. [S.l.]: MIT Press, 1995. 47, 48

ITO, K.; YODER, J. W.; WASHIZAKI, H.; FUKAZAWA, Y. A pattern language for knowledge handover when people transition. In: NOBLE, J.; JOHNSON, R.; ZDUN, U.; WALLINGFORD, E. (Ed.). **Transactions on pattern languages of programming IV**. [S.l.]: Springer, 2019. p. 183–209. 54, 55

IVARSSON, M.; GORSCHEK, T. A method for evaluating rigor and industrial relevance of technology evaluations. **Empirical Software Engineering**, v. 16, n. 3, p. 365–395, 2011. 11, 40

JANES, A.; LENARDUZZI, V.; STAN, A. C. A continuous software quality monitoring approach for small and medium enterprises. In: INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING COMPANION, 8., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 97–100. 33, 159, 164

JEONG, A. Comparing instructional event sequences in audio podcasts with low versus high user satisfaction. **TechTrends**, v. 63, n. 5, p. 559–563, 2019. 51

JIARPAKDEE, J.; TANTITHAMTHAVORN, C.; TREUDE, C. Autospearman: automatically mitigating correlated software metrics for interpreting defect models. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION, 2018. **Proceedings...** [S.l.]: IEEE, 2018. p. 92–103. 29, 161, 163

JOHNSON, P. M. Searching under the streetlight for useful software analytics. **IEEE Software**, v. 30, n. 4, p. 57–63, 2013. 33, 157, 164

JOYCE, A.; PAQUIN, R. L. The triple layered business model canvas: a tool to design more sustainable business models. **Journal of Cleaner Production**, v. 135, p. 1474–1486, 2016. 87

KARIM, M. R.; ALAM, A.; DIDAR, S.; KABEER, S. J.; RUHE, G.; BALUTA, B.; MAHMUD, S. Applying data analytics towards optimized issue management: an industrial case study. In: INTERNATIONAL WORKSHOP ON CONDUCTING EMPIRICAL STUDIES IN INDUSTRY, 4., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 7–13. 27, 159, 163

KARNA, H.; VICKOVIĆ, L.; GOTOVAC, S. Application of data mining methods for effort estimation of software projects. **Software: Practice and Experience**, v. 49, n. 2, p. 171–191, 2019. 29, 160, 163

KIDWELL, B.; HAYES, J. H. Toward a learned project-specific fault taxonomy: application of software analytics. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 2015. [S.l.], 2015. p. 1–4. 25, 158, 163

KIM, M.; ZIMMERMANN, T.; DELINE, R.; BEGEL, A. The emerging role of data scientists on software development teams. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 38., 2016. **Proceedings...** [S.l.]: IEEE/ACM, 2016. p. 96–107. 3, 35, 75, 159, 165

_____. Data scientists in software teams: state of the art and challenges. **IEEE Transactions on Software Engineering**, v. 44, n. 11, p. 1024–1038, 2018. 35, 161, 164

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing systematic literature reviews in software engineering**. [S.l.]: EBSE, 2007. 9, 11

KOCAGUNELI, E.; CUKIC, B.; LU, H. Predicting more from less: synergies of learning. In: INTERNATIONAL WORKSHOP ON REALIZING ARTIFICIAL INTELLIGENCE SYNERGIES IN SOFTWARE ENGINEERING, 2., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 42–48. 28, 157, 163

KOCAGUNELI, E.; MENZIES, T.; KEUNG, J.; COK, D.; MADACHY, R. Active learning and effort estimation: finding the essential content of software effort estimation data. **IEEE Transactions on Software Engineering**, v. 39, n. 8, p. 1040–1053, 2013. 29, 157, 163

KONDO, M.; BEZEMER, C.-P.; KAMEI, Y.; HASSAN, A. E.; MIZUNO, O. The impact of feature reduction techniques on defect prediction models. **Empirical Software Engineering**, p. 1–39, 2019. 30, 162, 163

KRISHNA, R. Learning effective changes for software projects. In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 32., 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 1002–1005. 25, 160, 163

KRISHNA, R.; MENZIES, T. Bellwethers: a baseline method for transfer learning. **IEEE Transactions on Software Engineering**, 2018. 30, 161, 163

KS, A. Ideation to production: can it be a one man show? **IEEE Engineering Management Review**, v. 45, n. 4, p. 28–29, 2017. 86

LEHMANN-WILLENBROCK, N.; ALLEN, J. A. Modeling temporal interaction dynamics in organizational settings. **Journal of Business and Psychology**, v. 33, n. 3, p. 325–344, 2018. 102

LEHTONEN, T.; ELORANTA, V.-P.; LEPPÄNEN, M.; ISOHANNI, E. Visualizations as a basis for agile software process improvement. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 20., 2013. **Proceedings...** [S.l.]: IEEE, 2013. v. 1, p. 495–502. 36, 157, 165

LI, Z.; AVGERIOU, P.; LIANG, P. A systematic mapping study on technical debt and its management. **Journal of Systems and Software**, v. 101, p. 193–220, 2015. 65

LICORISH, S. A.; TAHIR, A.; BOSU, M. F.; MACDONELL, S. G. On satisfying the android os community: user feedback still central to developers' portfolios. In: AUSTRALASIAN SOFTWARE ENGINEERING CONFERENCE, 24., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 78–87. 33, 158, 164

LIECHTI, O.; PASQUIER, J.; REIS, R. Beyond dashboards: on the many facets of metrics and feedback in agile organizations. In: INTERNATIONAL WORKSHOP ON COOPERATIVE AND HUMAN ASPECTS OF SOFTWARE ENGINEERING, 10., 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 16–22. 39, 86, 159, 165

______. Supporting agile teams with a test analytics platform: a case study. In: INTERNATIONAL WORKSHOP ON AUTOMATION OF SOFTWARE TESTING, 12., 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 9–15. 4, 38, 75, 125, 160, 165

LIN, J.; LIU, Y.; GUO, J.; CLELAND-HUANG, J.; GOSS, W.; LIU, W.; LOHAR, S.; MONAIKUL, N.; RASIN, A. Tiqi: a natural language interface for querying software project data. In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 32., 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 973–977. 27, 160, 163

LISMONT, J.; VANTHIENEN, J.; BAESENS, B.; LEMAHIEU, W. Defining analytics maturity indicators: a survey approach. **International Journal of Information Management**, v. 37, n. 3, p. 114–124, 2017. 1

LIU, F.; MAITLIS, S.; MILLS, A.; DUREPOS, G.; WIEBE, E. Nonparticipant observation. In: MILLS, A. J.; DUREPOS, G.; WIEBE, E. (Ed.). **Encyclopedia of case study research**. [S.l.]: SAGE, 2010. v. 2, p. 610–612. 46

LOU, J.-G.; LIN, Q.; DING, R.; FU, Q.; ZHANG, D.; XIE, T. Software analytics for incident management of online services: an experience report. In: INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 28., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 475–485. 25, 61, 157, 163

_____. Experience report on applying software analytics in incident management of online service. **Automated Software Engineering**, v. 24, n. 4, p. 905–941, 2017. 25, 160, 163

LOW, J. F.; YATHOG, T.; SVETINOVIC, D. Software analytics study of open-source system survivability through social contagion. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL ENGINEERING AND ENGINEERING MANAGEMENT, 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 1213–1217. 31, 158, 164

MAALEJ, W.; KURTANOVIC, Z.; NABIL, H.; STANIK, C. On the automatic classification of app reviews. **Requirements Engineering**, v. 21, n. 3, p. 311–331, 2016. 2, 26, 159, 163

MAALEJ, W.; NAYEBI, M.; JOHANN, T.; RUHE, G. Toward data-driven requirements engineering. **IEEE Software**, v. 33, n. 1, p. 48–54, 2016. 1, 25, 159, 163

MANZANO, M.; GÓMEZ, C.; AYALA, C.; MARTÍNEZ-FERNÁNDEZ, S.; RAM, P.; RODRÍGUEZ, P.; ORIOL, M. Definition of the on-time delivery indicator in rapid software development. In: INTERNATIONAL WORKSHOP ON QUALITY REQUIREMENTS IN AGILE PROJECTS, 1., 2018. **Proceedings...** [S.l.]: IEEE, 2018. p. 1–5. 34, 161, 164

MARTÍNEZ-FERNÁNDEZ, S.; JEDLITSCHKA, A.; GUZMÁN, L.; VOLLMER, A. M. A quality model for actionable analytics in rapid software development. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS, 44., 2018. **Proceedings...** [S.l.], 2018. p. 370–377. 35, 78, 160, 164

MATTILA, A.-L.; LEHTONEN, T.; TERHO, H.; MIKKONEN, T.; SYSTÄ, K. Mashing up software issue management, development, and usage data. In:

INTERNATIONAL WORKSHOP ON RAPID CONTINUOUS SOFTWARE ENGINEERING, 2., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 26–29. 37, 158, 165

MATTILA, A.-L.; SYSTÄ, K.; SIEVI-KORTE, O.; LEPPÄNEN, M.; MIKKONEN, T. Discovering software process deviations using visualizations. In: INTERNATIONAL CONFERENCE ON AGILE SOFTWARE DEVELOPMENT, 2017. **Proceedings...** [S.l.]: Springer, Cham, 2017. p. 259–266. 36, 159, 165

MAURYA, A. **Running lean: iterate from plan A to a plan that works**. [S.l.]: O'Reilly Media, 2012. 87

MCGRATH, S.; BASTOLA, K.; SIY, H. Concept to commit. In: INTERNATIONAL WORKSHOP ON DATA ANALYSIS PATTERNS IN SOFTWARE ENGINEERING, 1., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 6–8. 55

MCINTOSH, S.; NAGAPPAN, M.; ADAMS, B.; MOCKUS, A.; HASSAN, A. E. A large-scale empirical study of the relationship between build technology and build maintenance. **Empirical Software Engineering**, v. 20, n. 6, p. 1587–1633, 2015. 31, 158, 164

MELEGATI, J.; GOLDMAN, A. Seven patterns for software startups. In: CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 22., 2015. **Proceedings...** [S.l.]: Hillside Group, 2015. p. 20. 76

MENZIES, T. Predicting the future of predictive modeling. In: NSF WORKSHOP: PLANNING FUTURE DIRECTIONS IN AI &SE, 2012. **Proceedings...** [S.l.]: Citeseer, 2012. 28, 157, 163

______. Beyond data mining. **IEEE Software**, v. 30, n. 3, p. 92–92, 2013. 30, 157, 164

______. The unreasonable effectiveness of software analytics. **IEEE Software**, v. 35, n. 2, p. 96–98, 2018. 28, 161, 163

MENZIES, T.; ZIMMERMANN, T. Software analytics: so what? **IEEE Software**, v. 30, n. 4, p. 31–37, 2013. 10, 11, 16

______. Software analytics: what's next? **IEEE Software**, v. 35, n. 5, p. 64–70, 2018. 28, 161, 164

MERSON, P.; AGUIAR, A.; GUERRA, E.; YODER, J. Continuous inspection: a pattern for keeping your code healthy and aligned to the architecture. In: ASIAN CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 3., 2014. **Proceedings...** [S.l.], 2014. p. 6–8. 78

MEZOUAR, M. E.; ZHANG, F.; ZOU, Y. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. **Empirical Software Engineering**, v. 23, n. 3, p. 1704–1742, 2018. 32, 161, 164

MINELLI, R.; LANZA, M. Samoa-a visual software analytics platform for mobile applications. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 476–479. 67

______. Software analytics for mobile applications–insights & lessons learned. In: EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING, 17., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 144–153. 38, 157, 165

MINKU, L. L.; MENDES, E.; TURHAN, B. Data mining for software engineering and humans in the loop. **Progress in Artificial Intelligence**, v. 5, n. 4, p. 307–314, 2016. 28, 159, 163

MISIRLI, A. T.; CAGLAYAN, B.; BENER, A.; TURHAN, B. A retrospective study of software analytics projects: in-depth interviews with practitioners. **IEEE Software**, v. 30, n. 5, p. 54–61, 2013. 30, 157, 163

MORALES-RAMIREZ, I.; KIFETEW, F. M.; PERINI, A. Speech-acts based analysis for requirements discovery from online discussions. **Information Systems**, 2018. 26, 161, 163

MOSER, M.; PICHLER, J.; FLECK, G.; WITLATSCHIL, M. Rbg: a documentation generator for scientific and engineering software. In: INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION AND REENGINEERING, 22., 2015. **Proceedings...** [S.l.]: IEEE, 2015. p. 464–468. 32, 158, 164

MUSSON, R.; RICHARDS, J.; FISHER, D.; BIRD, C.; BUSSONE, B.; GANGULY, S. Leveraging the crowd: how 48,000 users helped improve lync performance. **IEEE Software**, v. 30, n. 4, p. 38–45, 2013. 3, 36, 157, 165

NAGLE, T.; SAMMON, D. The development of a design research canvas for data practitioners. **Journal of Decision Systems**, v. 25, n. sup1, p. 369–380, 2016. 87

NAYEBI, M.; KABEER, S. J.; RUHE, G.; CARLSON, C.; CHEW, F. Hybrid labels are the new measure! **IEEE Software**, v. 35, n. 1, p. 54–57, 2017. 28, 160, 163

NAYEBI, M.; RUHE, G.; ZIMMERMANN, T. Mining treatment-outcome constructs from sequential software engineering data. **IEEE Transactions on Software Engineering**, p. 1–20, 2019. 26, 162, 163

NERI, H. R.; TRAVASSOS, G. H. Measuresoftgram: a future vision of software product quality. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 12., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 54. 34, 161, 164

NGUYEN, D.; TIEN, D.; KESAVULU, M.; HELFERT, M. Usage analytics: research directions to discover insights from cloud-based applications. In: INTERNATIONAL CONFERENCE ON SMART CITIES AND GREEN ICT SYSTEMS, 7., 2018. **Proceedings...** [S.l.]: Sitepress, 2018. p. 1–8. 35, 161, 165

NIDAGUNDI, P.; NOVICKIS, L. Introducing lean canvas model adaptation in the scrum software testing. **Procedia Computer Science**, v. 104, p. 97–103, 2017. 87

NOEI, E.; ZHANG, F.; WANG, S.; ZOU, Y. Towards prioritizing user-related issue reports of mobile applications. **Empirical Software Engineering**, p. 1–33, 2019. 32, 162, 164

NOORWALI, I. Stakeholder concern-driven requirements analytics. **ACM Software Engineering Notes**, v. 43, n. 1, p. 1–6, 2018. 37, 161, 165

NORD, R. L.; OZKAYA, I.; KOZIOLEK, H.; AVGERIOU, P. Quantifying software architecture quality report on the first international workshop on software architecture metrics. **ACM Software Engineering Notes**, v. 39, n. 5, p. 32–34, 2014. 62

OLIVEIRA, R. de M.; GOLDMAN, A.; MELO, C. O. Designing and managing agile informative workspaces: discovering and exploring patterns. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCE, 46., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 4790–4799. 85

OLSON, G. M.; HERBSLEB, J. D.; REUTER, H. H. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. **Human–Computer Interaction**, v. 9, n. 3-4, p. 427–472, 1994. 50, 109

ONOUE, S.; HATA, H.; MONDEN, A.; MATSUMOTO, K. Investigating and projecting population structures in open source software projects: a case study of projects in github. **Transactions on Information and Systems**, v. 99, n. 5, p. 1304–1315, 2016. 27, 159, 163

OSTERWALDER, A.; PIGNEUR, Y. **Business model generation: a handbook for visionaries, game changers, and challengers**. [S.l.]: John Wiley & Sons, 2010. 87

OSTERWALDER, A.; PIGNEUR, Y.; TUCCI, C. L. Clarifying business models: origins, present, and future of the concept. **Communications of the Association for Information Systems**, v. 16, n. 1, p. 1, 2005. 87

PACHIDI, S.; SPRUIT, M.; WEERD, I. V. D. Understanding users' behavior with software operation data mining. **Computers in Human Behavior**, v. 30, p. 583–594, 2014. 3, 64

PETERS, F. On privacy and utility while improving software quality. In: INTERNATIONAL CONFERENCE ON CURRENT TRENDS IN THEORY AND PRACTICE OF COMPUTER SCIENCE, 43., 2018. **Proceedings...** [S.l.], 2018. p. 14. 28, 161, 163

PETERSEN, K. Measuring and predicting software productivity: a systematic map and review. **Information and Software Technology**, v. 53, n. 4, p. 317–343, 2011. 15

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 12., 2008. **Proceedings...** [S.l.], 2008. p. 1–10. 15

PETERSEN, K.; GENCEL, C. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In: INTERNATIONAL CONFERENCE ON SOFTWARE PROCESS AND PRODUCT MEASUREMENT, 8., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 81–89. 111, 112

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: an update. **Information and Software Technology**, v. 64, p. 1–18, 2015. 9, 13

PICCIONI, M.; FURIA, C. A.; MEYER, B. An empirical study of api usability. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 2013. **Proceedings...** [S.l.]: ACM, 2013. p. 5–14. 5

PINTO, A. F.; FONTES, N.; GUERRA, E.; TERRA, R. Archci: an architectural verification tool into continuous integration. In: BRAZILIAN CONFERENCE ON SOFTWARE: THEORY AND PRACTICE, 11., 2016. **Proceedings...** [S.l.], 2016. p. 121–128. 70

POHL, M.; WALLNER, G.; KRIGLSTEIN, S. Using lag-sequential analysis for understanding interaction sequences in visualizations. **International Journal of Human-Computer Studies**, v. 96, p. 54–66, 2016. 50

PORT, D.; TABER, B. Actionable analytics for strategic maintenance of critical software: an industry experience report. **IEEE Software**, v. 35, n. 1, p. 58–63, 2018. 35, 160, 165

PRICE, P. C. et al. **Research methods in psychology**. [S.l.]: BCCampus, 2015. 46

PURAO, S. **Design research in the technology of information systems: Truth or dare**. [S.l.]: GSU Department of CIS, 2002. 45–77 p. 43

RAHMAN, A.; AGRAWAL, A.; KRISHNA, R.; SOBRAN, A. Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 4., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 8–14. 26, 161, 163

RAM, P.; RODRIGUEZ, P.; OIVO, M. Software process measurement and related challenges in agile software development: a multiple case study. In: INTERNATIONAL CONFERENCE ON PRODUCT-FOCUSED SOFTWARE PROCESS IMPROVEMENT, 2018. **Proceedings...** [S.l.]: Springer, 2018. p. 272–287. 81

RAM, P.; RODRIGUEZ, P.; OIVO, M.; MARTÍNEZ-FERNÁNDEZ, S. Success factors for effective process metrics operationalization in agile software development: a multiple case study. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM PROCESSES, 2019. **Proceedings...** [S.l.]: IEEE, 2019. p. 14–23. 86

RAMARAO, P.; MUTHUKUMARAN, K.; DASH, S.; MURTHY, N. B. Impact of bug reporter's reputation on bug-fix times. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS ENGINEERING, 2016. **Proceedings...** [S.l.]: IEEE, 2016. p. 57–61. 3

REDDIVARI, S.; RAD, S.; BHOWMIK, T.; CAIN, N.; NIU, N. Visual requirements analytics: a framework and case study. **Requirements Engineering**, v. 19, n. 3, p. 257–279, 2014. 37, 158, 165

RISING, L. Patterns mining. In: ZAMIR, S. (Ed.). **Handbook of object technology**. [S.l.]: CRC Press, 1999. p. 38–31. 54

_____. Small experiments. **Better Software**, n. Jan-Feb, p. 13–44, 2011. 74

ROBBES, R.; VIDAL, R.; BASTARRICA, M. C. Are software analytics efforts worthwhile for small companies? the case of amisoft. **IEEE Software**, v. 30, n. 5, p. 46–53, 2013. 3, 4, 35, 157, 164

ROBILLARD, P. N.; D'ASTOUS, P.; DÉTIENNE, F.; VISSER, W. Measuring cognitive activities in software engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 20., 1998. **Proceedings...** [S.l.]: IEEE, 1998. p. 292–300. 47

ROBLES, G.; GONZÁLEZ-BARAHONA, J. M. Mining student repositories to gain learning analytics: an experience report. In: GLOBAL ENGINEERING EDUCATION CONFERENCE, 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 1249–1254. 31, 157, 164

ROBLES, G.; GONZÁLEZ-BARAHONA, J. M.; CERVIGÓN, C.; CAPILUPPI, A.; IZQUIERDO-CORTÁZAR, D. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In: WORKING CONFERENCE ON MINING SOFTWARE REPOSITORIES, 11., 2014. **Proceedings...** [S.l.]: ACM, 2014. p. 222–231. 62

ROSEN, C.; GRAWI, B.; SHIHAB, E. Commit guru: analytics and risk prediction of software commits. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 10., 2015. **Proceedings...** [S.l.]: ACM, 2015. p. 966–969. 26, 158, 163

RUNESON, P.; HÖST, M.; RAINER, A.; REGNELL, B. **Case study research in software engineering: guidelines and examples**. [S.l.]: John Wiley and Sons, 2012. 13, 46

SANDERSON, P. M.; FISHER, C. Exploratory sequential data analysis: foundations. **Human–Computer Interaction**, v. 9, n. 3-4, p. 251–317, 1994. 47, 50, 52, 125, 127

SANT'ANNA, N.; GUERRA, E.; IVO, A.; PEREIRA, F.; MORAES, M.; GOMES, V.; VERAS, L. G. Modelo arquitetural para coleta, processamento e visualização de informações de clima espacial. In: SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 10., 2014. **Anais...** [S.l.]: SBC, 2014. p. 125–136. 97

SCAIFE, M.; ROGERS, Y. External cognition: how do graphical representations work? **International Journal of Human-Computer Studies**, v. 45, n. 2, p. 185–213, 1996. 49

SHARP, H.; GIUFFRIDA, R.; MELNIK, G. Information flow within a dispersed agile team: a distributed cognition perspective. In: INTERNATIONAL CONFERENCE ON AGILE SOFTWARE DEVELOPMENT, 2012. **Proceedings...** [S.l.]: Springer, 2012. p. 62–76. 48

SHARP, H.; ROBINSON, H. A distributed cognition account of mature xp teams. In: INTERNATIONAL CONFERENCE ON EXTREME PROGRAMMING AND AGILE PROCESSES IN SOFTWARE ENGINEERING, 2006. **Proceedings...** [S.l.]: Springer, 2006. p. 1–10. 48

SHARP, H.; ROBINSON, H.; SEGAL, J.; FURNISS, D. The role of story cards and the wall in xp teams: a distributed cognition perspective. In: AGILE CONFERENCE, 2006. **Proceedings...** [S.l.]: IEEE, 2006. 85

SHULL, F. Data, data everywhere. **IEEE Software**, v. 31, n. 5, 2014. 2, 4, 24, 81, 158, 163

SIMON, H. A. **The sciences of the artificial**. [S.l.]: MIT Press, 1996. 43

SINGH, M.; WALIA, G. S.; GOSWAMI, A. An empirical investigation to overcome class-imbalance in inspection reviews. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING AND DATA SCIENCE, 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 15–22. 33, 159, 164

SMITH, S.; DUKE, D.; WRIGHT, P. Using the resources model in virtual environment design. In: WORKSHOP ON USER CENTERED DESIGN AND IMPLEMENTATION OF VIRTUAL ENVIRONMENTS, 1999. **Proceedings...** [S.l.], 1999. p. 57–72. 50

SNYDER, B.; CURTIS, B. Using analytics to guide improvement during an agile–devops transformation. **IEEE Software**, v. 35, n. 1, p. 78–83, 2018. 40, 42, 161, 165

SOLTANIFAR, B.; AKBARINASAJI, S.; CAGLAYAN, B.; BENER, A. B.; FILIZ, A.; KRAMER, B. M. Software analytics in practice: a defect prediction model using code smells. In: INTERNATIONAL DATABASE ENGINEERING & APPLICATIONS SYMPOSIUM, 20., 2016. **Proceedings...** [S.l.], 2016. p. 148–155. 29, 159, 163

SOUZA, R.; CHAVEZ, C.; BITTENCOURT, R. Patterns for cleaning up bug data. In: INTERNATIONAL WORKSHOP ON DATA ANALYSIS PATTERNS IN SOFTWARE ENGINEERING, 1., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 26–28. 55

SOUZA, R.; CHAVEZ, C.; BITTENCOURT, R. A. Rapid releases and patch backouts: a software analytics approach. **IEEE Software**, v. 32, n. 2, p. 89–96, 2015. 26, 72, 158, 163

STOREY, M.-A. Lies, damned lies, and analytics: why big data needs thick data. In: MENZIES, T.; WILLIAMS, L.; ZIMMERMANN, T. (Ed.). **Perspectives on data science for software engineering**. [S.l.]: Elsevier, 2016. p. 369–374. 2

SUONSYRJÄ, S.; MIKKONEN, T. Designing an unobtrusive analytics framework for monitoring java applications. In: KOBYLINSKI, A.; CZARNACKA-CHROBOT, B.; SWIERCZEK, J. (Ed.). **Software measurement**. [S.l.]: Springer, 2015. p. 160–175. 33, 158, 164

SUONSYRJÄ, S.; SYSTÄ, K.; MIKKONEN, T.; TERHO, H. Collecting usage data for software development: selection framework for technological approaches. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, 28., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 114–119. 33, 65, 159, 164

SUREKA, A.; SINGH, H. K.; BAGEWADI, M.; MITRA, A.; KARANTH, R. A decision support platform for guiding a bug triager for resolver recommendation using textual and non-textual features. In: INTERNATIONAL WORKSHOP ON QUANTITATIVE APPROACHES TO SOFTWARE QUALITY, 3., 2015. **Proceedings...** [S.l.], 2015. p. 25. 3, 39, 158, 165

SYED-MOHAMAD, S. M.; HARON, N. H.; MCBRIDE, T. Test adequacy assessment using test-defect coverage analytic model. **Journal of**

Telecommunication, Electronic and Computer Engineering (JTEC), v. 9, n. 3-5, p. 191–196, 2017. 34, 160, 164

TAIPALE, T.; QVIST, M.; TURHAN, B. Constructing defect predictors and communicating the outcomes to practitioners. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 2013. **Proceedings...** [S.l.]: ACM, 2013. p. 357–362. 29, 67, 157, 163

TAMLA, P.; FEJA, S.; PRAUSE, C. R. Metadata-based code example embedding. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 3., 2017. **Proceedings...** [S.l.]: ACM, 2017. p. 15–18. 31, 160, 164

TANTITHAMTHAVORN, C.; HASSAN, A. E. An experience report on defect modelling in practice: pitfalls and challenges. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING: SOFTWARE ENGINEERING IN PRACTICE, 40., 2018. **Proceedings...** [S.l.], 2018. p. 286–295. 30, 160, 163

THEODOROU, V.; GEROSTATHOPOULOS, I.; AMINI, S.; SCANDARIATO, R.; PREHOFER, C.; STARON, M. Theta architecture: preserving the quality of analytics in data-driven systems. In: EUROPEAN CONFERENCE ON ADVANCES IN DATABASES AND INFORMATION SYSTEMS, 2017. **Proceedings...** [S.l.]: Springer, 2017. p. 186–198. 25, 160, 163

TORRES, A. G.; GARCÍA-PEÑALVO, F. J.; THERÓN-SÁNCHEZ, R. How evolutionary visual software analytics supports knowledge discovery. **Journal of Information Science and Engineering**, v. 29, n. 1, p. 17–34, 2013. 37, 157, 165

TRAUTSCH, F.; HERBOLD, S.; MAKEDONSKI, P.; GRABOWSKI, J. Adressing problems with external validity of repository mining studies through a smart data platform. In: INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES, 13., 2016. **Proceedings...** [S.l.]: ACM, 2016. p. 97–108. 38, 159, 165

______. Addressing problems with replicability and validity of repository mining studies through a smart data platform. **Empirical Software Engineering**, v. 23, n. 2, p. 1036–1083, 2018. 38, 160, 165

TREUDE, C.; FIGUEIRA-FILHO, F.; KULESZA, U. Summarizing and measuring development activity. In: JOINT MEETING ON FOUNDATIONS OF

SOFTWARE ENGINEERING, 10., 2015. **Proceedings...** [S.l.]: ACM, 2015. p. 625–636. 34, 158, 164

TURHAN, B.; KUUTTI, K. Simpler questions can lead to better insights. In: MENZIES, T.; WILLIAMS, L.; ZIMMERMANN, T. (Ed.). **Perspectives on data science for software engineering**. Boston: Morgan Kaufmann, 2016. 70

VARGAS, E. L.; HEJDERUP, J.; KECHAGIA, M.; BRUNTINK, M.; GOUSIOS, G. Enabling real-time feedback in software engineering. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 40., 2018. **Proceedings...** [S.l.]: ACM, 2018. p. 21–24. 38, 161, 165

VENABLE, J.; PRIES-HEJE, J.; BASKERVILLE, R. Feds: a framework for evaluation in design science research. **European Journal of Information Systems**, v. 25, n. 1, p. 77–89, 2016. 94

VICK, R. M. et al. Assessment of resource coordination effectiveness through analysis of distributed cognitive traces in team decision making. In: ANNUAL MEETING OF THE COGNITIVE SCIENCE SOCIETY, 25., 2003. **Proceedings...** [S.l.], 2003. 49

WANG, C.; AKELLA, R.; RAMACHANDRAN, S.; HINNANT, D. Knowledge extraction and reuse within "smart" service centers. In: ANNUAL SRII GLOBAL CONFERENCE, 2011. **Proceedings...** [S.l.]: IEEE, 2011. p. 163–176. 3

WEISS, D. M.; MOCKUS, A. The chunking pattern. In: INTERNATIONAL WORKSHOP ON DATA ANALYSIS PATTERNS IN SOFTWARE ENGINEERING, 1., 2013. **Proceedings...** [S.l.]: IEEE, 2013. p. 35–37. 72

WIERINGA, R.; MAIDEN, N.; MEAD, N.; ROLLAND, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. **Requirements Engineering**, v. 11, n. 1, p. 102–107, 2006. 13

WILLIAMS, J.; MATRAGKAS, N.; KOLOVOS, D.; KORKONTZELOS, I.; ANANIADOU, S.; PAIGE, R. Software analytics for mde communities. In: WORKSHOP ON OPEN SOURCE SOFTWARE FOR MODEL DRIVEN ENGINEERING, 1. [S.l.], 2014. p. 53–63. 31, 158, 164

WIRFS-BROCK, R.; YODER, J.; GUERRA, E. Patterns to develop and evolve architecture during an agile software project. In: CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 22., 2015. **Proceedings...** [S.l.]: Hillside Group, 2015. p. 9. 82

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 18., 2014. **Proceedings...** [S.l.]: ACM, 2014. p. 38. 11

WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.: s.n.], 2012. 46

WRIGHT, P. C.; FIELDS, R. E.; HARRISON, M. D. Analyzing human-computer interaction as distributed cognition: the resources model. **Human-Computer Interaction**, v. 15, n. 1, p. 1–41, 2000. 47, 48, 49, 52, 100, 101, 104, 111, 125, 127

WU, R. Diagnose crashing faults on production software. In: INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 22., 2014. **Proceedings...** [S.l.]: ACM, 2014. p. 771–774. 31, 158, 164

WU, R.; WEN, M.; CHEUNG, S.-C.; ZHANG, H. Changelocator: locate crash-inducing changes based on crash reports. **Empirical Software Engineering**, v. 23, n. 5, p. 2866–2900, 2018. 31, 161, 164

WU, R.; ZHANG, H.; CHEUNG, S.-C.; KIM, S. Crashlocator: locating crashing faults based on crash stacks. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE TESTING AND ANALYSIS, 2014. **Proceedings...** [S.l.], 2014. p. 204–214. 31, 158, 164

YE, X.; BUNESCU, R.; LIU, C. Learning to rank relevant files for bug reports using domain knowledge. In: INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, 22., 2014. **Proceedings...** [S.l.]: ACM, 2014. p. 689–699. 31, 158, 164

YODER, J.; WIRFS-BROCK, R. Qa to aq part two: shifting from quality assurance to agile quality. In: CONFERENCE ON PATTERNS OF PROGRAMMING LANGUAGE, 21., 2014. **Proceedings...** [S.l.], 2014. 62, 65, 67, 78, 82

YUZUKI, R.; HATA, H.; MATSUMOTO, K. How we resolve conflict: an empirical study of method-level conflict resolution. In: INTERNATIONAL WORKSHOP ON SOFTWARE ANALYTICS, 1., 2015. **Proceedings...** [S.l.], 2015. p. 21–24. 33, 158, 164

ZHANG, D.; DANG, Y.; LOU, J.-G.; HAN, S.; ZHANG, H.; XIE, T. Software analytics as a learning case in practice: approaches and experiences. In:

INTERNATIONAL WORKSHOP ON MACHINE LEARNING TECHNOLOGIES IN SOFTWARE ENGINEERING, 2011. **Proceedings...** [S.l.]: ACM, 2011. p. 55–58. 1, 3, 10, 11, 16

ZHANG, D.; HAN, S.; DANG, Y.; LOU, J.-G.; ZHANG, H.; XIE, T. Software analytics in practice. **IEEE Software**, v. 30, n. 5, p. 30–37, 2013. 39, 42, 157, 165

ZHOU, X.; JIN, Y.; ZHANG, H.; LI, S.; HUANG, X. A map of threats to validity of systematic literature reviews in software engineering. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 23., 2016. **Proceedings...** [S.l.]: IEEE, 2016. p. 153–160. 41

ZHU, J.; HE, P.; XIE, Q.; ZHENG, Z.; LYU, M. R. Carp: context-aware reliability prediction of black-box web services. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, 2017. **Proceedings...** [S.l.]: IEEE, 2017. p. 17–24. 28, 159, 163

APPENDIX A - LIST OF PAPERS OF THE SYSTEMATIC MAPPING STUDY

Table A.1 - List of selected studies.

#	Title	Reference	Year
S1	Predicting the Future of Predictive Modeling	(MENZIES, 2012)	2012
S2	Software mining and fault prediction	(CATAL, 2012)	2012
S3	SQuORE: A new approach to software project quality measurement	(BALDASSARI, 2012)	2012
S4	A Framework for the Evolutionary Visual Software Analytics Process	(GONZÁLEZ-TORRES et al., 2013a)	2013
S5	A goal driven framework for software project data analytics	(CHATZIKONSTANTINOU et al., 2013)	2013
S6	A Retrospective Study of Software Analytics Projects: In-Depth Interviews with Practitioners	(MISIRLI et al., 2013)	2013
S7	Active learning and effort estimation: Finding the essential content of software effort estimation data	(KOCAGUNELI et al., 2013b)	2013
S8	Are Software Analytics Efforts Worthwhile for Small Companies? The Case of Amisoft	(ROBBES et al., 2013)	2013
S9	Beyond data mining	(MENZIES, 2013)	2013
S10	CODEMINE: Building a Software Development Data Analytics Platform at Microsoft	(CZERWONKA et al., 2013)	2013
S11	Constructing defect predictors and communicating the outcomes to practitioners	(TAIPALE et al., 2013)	2013
S12	Developer Dashboards: The Need for Qualitative Analytics	(BAYSAL et al., 2013a)	2013
S13	How Evolutionary Visual Software Analytics Supports Knowledge Discovery (similar)	(TORRES et al., 2013)	2013
S14	Informing development decisions: From data to information	(BAYSAL, 2013)	2013
S15	Leveraging the Crowd: How 48,000 Users Helped Improve Lync Performance	(MUSSON et al., 2013)	2013
S16	Mining student repositories to gain learning analytics. An experience report	(ROBLES; GONZÁLEZ-BARAHONA, 2013)	2013
S17	Predicting more from less: Synergies of learning	(KOCAGUNELI et al., 2013a)	2013
S18	Searching under the Streetlight for Useful Software Analytics	(JOHNSON, 2013)	2013
S19	Software analytics for incident management of online services: An experience report	(LOU et al., 2013)	2013
S20	Software Analytics for Mobile Applications–Insights & Lessons Learned	(MINELLI; LANZA, 2013b)	2013
S21	Software Analytics in Practice	(ZHANG et al., 2013)	2013
S22	Supporting software decision meetings: Heatmaps for visualising test and code measurements	(FELDT et al., 2013)	2013
S23	Understanding how companies interact with free software communities	(GONZALEZ-BARAHONA et al., 2013)	2013
S24	Using change entries to collect software project information	(ASUNCION et al., 2013)	2013
S25	Visualizations as a basis for agile software process improvement	(LEHTONEN et al., 2013)	2013
S26	An initial quality analysis of the Ohloh software evolution data	(BRUNTINK, 2014)	2014

continue

Table A.1 – *Continuation.*

#	Title	Reference	Year
S27	Analyze this! 145 questions for data scientists in software engineering	(BEGEL; ZIMMERMANN, 2014)	2014
S28	CrashLocator: Locating Crashing Faults Based on Crash Stacks	(WU et al., 2014)	2014
S29	Data stream mining for predicting software build outcomes using source code metrics	(FINLAY et al., 2014)	2014
S30	Data, Data Everywhere...	(SHULL, 2014)	2014
S31	Diagnose Crashing Faults on Production Software	(WU, 2014)	2014
S32	Learning to rank relevant files for bug reports using domain knowledge	(YE et al., 2014)	2014
S33	Software analytics for MDE communities	(WILLIAMS et al., 2014)	2014
S34	Visual requirements analytics: a framework and case study	(REDDIVARI et al., 2014)	2014
S35	A decision support platform for guiding a bug trigger for resolver recommendation using textual and non-textual features	(SUREKA et al., 2015)	2015
S36	A Large-Scale Empirical Study of the Relationship between Build Technology and Build Maintenance	(MCINTOSH et al., 2015)	2015
S37	Big data analytics on large-scale socio-technical software engineering archives	(BAYATI et al., 2015)	2015
S38	Commit Guru: Analytics and Risk Prediction of Software Commits	(ROSEN et al., 2015)	2015
S39	Designing an unobtrusive analytics framework for monitoring Java applications	(SUONSYRJÄ; MIKKONEN, 2015)	2015
S40	Do topics make sense to managers and developers?	(HINDLE et al., 2015)	2015
S41	How we resolve conflict: an empirical study of method-level conflict resolution	(YUZUKI et al., 2015)	2015
S42	Irish: A Hidden Markov Model to detect coded information islands in free text	(CERULO et al., 2015)	2015
S43	Mashing Up Software Issue Management, Development, and Usage Data	(MATTILA et al., 2015)	2015
S44	On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios	(LICORISH et al., 2015)	2015
S45	Rapid releases and patch backouts: A software analytics approach	(SOUZA et al., 2015)	2015
S46	RbG: A documentation generator for scientific and engineering software	(MOSER et al., 2015)	2015
S47	Software analytics study of Open-Source system survivability through social contagion	(LOW et al., 2015)	2015
S48	Summarizing and measuring development activity	(TREUDE et al., 2015)	2015
S49	Test and Defect Coverage Analytics Model for the assessment of software test adequacy	(HARON; SYED-MOHAMAD, 2015)	2015
S50	Toward a Learned Project-Specific Fault Taxonomy: Application of Software Analytics	(KIDWELL; HAYES, 2015)	2015
S51	Towards base rates in software analytics Early results and challenges from studying Ohloh	(BRUNTINK, 2015)	2015
S52	Understanding software artifact provenance	(GODFREY, 2015)	2015
S53	Why statically estimate code coverage is so hard? a report of lessons learned	(ANICHE et al., 2015)	2015

continue

Table A.1 – *Continuation.*

#	Title	Reference	Year
S54	A Hybrid Model for Task Completion Effort Estimation	(DEHGHAN et al., 2016)	2016
S55	Addressing Problems with External Validity of Repository Mining Studies Through a Smart Data Platform (SmartSHARK)	(TRAUTSCH et al., 2016)	2016
S56	Applying data analytics towards optimized issue management: an industrial case study	(KARIM et al., 2016)	2016
S57	Cold-start Software Analytics	(GUO et al., 2016)	2016
S58	Collecting Usage Data for Software Development: Selection Framework for Technological Approaches.	(SUONSYRJÄ et al., 2016)	2016
S59	Data mining for software engineering and humans in the loop	(MINKU et al., 2016)	2016
S60	Data Science Challenges to Improve Quality Assurance of Internet of Things Applications	(FOIDL; FELDERER, 2016)	2016
S61	DeepSoft: A Vision for a Deep Model of Software	(DAM et al., 2016)	2016
S62	Developer Targeted Analytics: Supporting Software Development Decisions with Runtime Information	(CITO, 2016)	2016
S63	Investigating and Projecting Population Structures in Open Source Software Projects: A Case Study of Projects in GitHub	(ONOUE et al., 2016)	2016
S64	Issues on Software Quality Models for Mastering Change	(FELDERER, 2016)	2016
S65	Knowledge discovery in software teams by means of evolutionary visual software analytics	(GONZÁLEZ-TORRES et al., 2016)	2016
S66	On the automatic classification of app reviews	(MAALEJ et al., 2016a)	2016
S67	Predicting delays in software projects using networked classification	(CHOETKIERTIKUL et al., 2015)	2016
S68	Quality-Impact Assessment of Software Systems	(FOTROUSI, 2016)	2016
S69	Software Analytics in Practice: A Defect Prediction Model Using Code Smells	(SOLTANIFAR et al., 2016)	2016
S70	Software Crowdsourcing Reliability: An Empirical Study on Developers Behavior	(ALELYANI; YANG, 2016)	2016
S71	Streaming Software Analytics	(GOUSIOS et al., 2016)	2016
S72	The Emerging Role of Data Scientists on Software Development Teams	(KIM et al., 2016)	2016
S73	The bones of the system: A case study of logging and telemetry at Microsoft	(BARIK et al., 2016)	2016
S74	Toward Data-Driven Requirements Engineering	(MAALEJ et al., 2016b)	2016
S75	A continuous software quality monitoring approach for small and medium enterprises	(JANES et al., 2017)	2017
S76	An Empirical Investigation to Overcome Class-imbalance in Inspection Reviews	(SINGH et al., 2017)	2017
S77	Beyond dashboards: on the many facets of metrics and feedback in agile organizations	(LIECHTI et al., 2017a)	2017
S78	CARP: Context-Aware Reliability Prediction of Black-Box Web Services	(ZHU et al., 2017)	2017
S79	Context-Based Analytics - Establishing Explicit Links between Runtime Traces and Source Code	(CITO et al., 2017)	2017
S80	Discovering Software Process Deviations Using Visualizations	(MATTILA et al., 2017)	2017
S81	Easy over Hard: A Case Study on Deep Learning	(FU; MENZIES, 2017)	2017

continue

Table A.1 – *Continuation.*

#	Title	Reference	Year
S82	Experience report on applying software analytics in incident management of online service (S19 extension)	(LOU et al., 2017)	2017
S83	Find, Understand, and Extend Development Screencasts on YouTube	(ELLMANN et al., 2017)	2017
S84	Hybrid Labels Are the New Measure!	(NAYEBI et al., 2017)	2017
S85	Learning effective changes for software projects	(KRISHNA, 2017)	2017
S86	Metadata-based Code Example Embedding	(TAMLA et al., 2017)	2017
S87	On the Similarity of Software Development Documentation	(ELLMANN, 2017)	2017
S88	Predicting Rankings of Software Verification Tools	(CZECH et al., 2017)	2017
S89	Strong Agile Metrics: Mining Log Data to Determine Predictive Power of Software Metrics for Continuous Delivery Teams	(HUIJGENS et al., 2017)	2017
S90	Supporting Agile Teams with a Test Analytics Platform: a Case Study (Beyond Dashboards)	(LIECHTI et al., 2017b)	2017
S91	Test adequacy assessment using test-defect coverage analytic model (S49)	(SYED-MOHAMAD et al., 2017)	2017
S92	Theta Architecture: Preserving the Quality of Analytics in Data-Driven Systems	(THEODOROU et al., 2017)	2017
S93	Timezone and Time-of-day Variance in GitHub Teams: An Empirical Method and Study	(DEVANBU et al., 2017)	2017
S94	TiQi: A natural language interface for querying software project data	(LIN et al., 2017)	2017
S95	Transferring code-clone detection and analysis to practice	(DANG et al., 2017)	2017
S96	(No) Influence of Continuous Integration on the Commit Activity in GitHub Projects	(BALTES et al., 2018)	2018
S97	Graal: The Quest for Source Code Knowledge	(COSENTINO et al., 2018)	2018
S98	A deep learning model for estimating story points	(CHOETKIERTIKUL et al., 2018)	2018
S99	A Proposal towards the Design of an Architecture for Evolutionary Visual Software Analytics	(GONZÁLEZ-TORRES et al., 2018)	2018
S100	A Quality Model for Actionable Analytics in Rapid Software Development	(MARTÍNEZ-FERNÁNDEZ et al., 2018)	2018
S101	Actionable Analytics for Strategic Maintenance of Critical Software: An Industry Experience Report	(PORT; TABER, 2018)	2018
S102	Addressing problems with replicability and validity of repository mining studies through a smart data platform	(TRAUTSCH et al., 2018)	2018
S103	An empirical study of crash-inducing commits in Mozilla Firefox	(AN et al., 2018)	2018
S104	An empirical study on the issue reports with questions raised during the issue resolving process	(HUANG et al., 2018b)	2018
S105	An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges	(TANTITHAMTHAVORN; HASSAN, 2018)	2018
S106	An investigation of the fault-proneness of clone evolutionary patterns	(BARBOUR et al., 2018)	2018
S107	Analyzing a decade of Linux system calls	(BAGHERZADEH et al., 2018)	2018
S108	Application of data mining methods for effort estimation of software projects	(KARNA et al., 2019)	2018

continue

Table A.1 – *Continuation.*

#	Title	Reference	Year
S109	Are tweets useful in the bug fixing process? An empirical study on Firefox and Chrome	(MEZOUAR et al., 2018)	2018
S110	Autospearman: Automatically mitigating correlated software metrics for interpreting defect models	(JIARPAKDEE et al., 2018)	2018
S111	Bellwethers: A Baseline Method For Transfer Learning	(KRISHNA; MENZIES, 2018)	2018
S112	Big Data and software engineering: prospects for mutual enrichment	(ARNDT, 2018)	2018
S113	Big time': An examination of temporal complexity and business value in analytics	(CONBOY et al., 2018)	2018
S114	ChangeLocator: locate crash-inducing changes based on crash reports	(WU et al., 2018)	2018
S115	Characterizing the Influence of Continuous Integration: Empirical Results from 250 Open Source and Proprietary Projects	(RAHMAN et al., 2018)	2018
S116	Data scientists in software teams: State of the art and challenges	(KIM et al., 2018)	2018
S117	Definition of the On-time Delivery Indicator in Rapid Software Development	(MANZANO et al., 2018)	2018
S118	Deploying Software Team Analytics in a Multinational Organization	(AUGUSTINE et al., 2018)	2018
S119	Enabling Real-Time Feedback in Software Engineering	(VARGAS et al., 2018)	2018
S120	Explainable Software Analytics	(DAM et al., 2018)	2018
S121	Facilitating Feasibility Analysis: The Pilot Defects Prediction Dataset Maker	(FALESSI; MOEDE, 2018)	2018
S122	Inference of development activities from interaction with uninstrumented applications	(BAO et al., 2018)	2018
S123	Is "Better Data" Better Than "Better Data Miners"?	(AGRAWAL et al., 2018)	2018
S124	Measuresoftgram: A Future Vision of Software Product Quality	(NERI; TRAVASSOS, 2018)	2018
S125	On Privacy and Utility while Improving Software Quality	(PETERS, 2018)	2018
S126	Perceval: Software Project Data at Your Will	(DUEÑAS et al., 2018)	2018
S127	Predicting failures in agile software development through data analytics	(BATARSEH; GONZALEZ, 2018)	2018
S128	Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction	(HUANG et al., 2018a)	2018
S129	Software analytics in continuous delivery: a case study on success factors	(HUIJGENS et al., 2018)	2018
S130	Software Analytics: What's Next?	(MENZIES; ZIMMERMANN, 2018)	2018
S131	Speech-acts based analysis for requirements discovery from online discussions	(MORALES-RAMIREZ et al., 2018)	2018
S132	Stakeholder Concern-Driven Requirements Analytics (doctoral research in progress)	(NOORWALI, 2018)	2018
S133	The Unreasonable Effectiveness of Software Analytics	(MENZIES, 2018)	2018
S134	Usage analytics: research directions to discover insights from cloud-based applications	(NGUYEN et al., 2018)	2018
S135	Using Analytics to Guide Improvement During an Agile/DevOps Transformation	(SNYDER; CURTIS, 2018)	2018

continue

Table A.1 – *Continuation.*

#	**Title**	**Reference**	**Year**
S136	Utilizing Product Usage Data for Requirements Evaluation	(HEMMATI et al., 2018)	2018
S137	An empirical comparison of dependency network evolution in seven software packaging ecosystems	(DECAN et al., 2019)	2019
S138	An empirical study of DLL injection bugs in the Firefox ecosystem	(AN et al., 2019)	2019
S139	Mining Treatment-Outcome Constructs from Sequential Software Engineering Data	(NAYEBI et al., 2019)	2019
S140	The impact of feature reduction techniques on defect prediction models	(KONDO et al., 2019)	2019
S141	Towards prioritizing user-related issue reports of mobile applications	(NOEI et al., 2019)	2019

SOURCE: Prepared by the author.

Conclusion.

APPENDIX B - TYPICAL ISSUES ADDRESSED TO SOFTWARE ANALYTICS

Data Analytics - Techniques, Methods and Tools (N=32)
Issues: software traceability (ASUNCION et al., 2013); project management (CHATZIKONSTANTINOU et al., 2013); teamwork and collaboration (GONZALEZ-BARAHONA et al., 2013) (KARIM et al., 2016) (LIN et al., 2017); service incident diagnosis (LOU et al., 2013) (LOU et al., 2017); software measurement (SHULL, 2014); data quality (BRUNTINK, 2014); fault taxonomy (KIDWELL; HAYES, 2015); rapid releases and code integration (SOUZA et al., 2015); risky software commits (ROSEN et al., 2015); data driven requirements engineering (MAALEJ et al., 2016b) (MAALEJ et al., 2016a); configuration of cold-start projects (GUO et al., 2016); quality requirements (FOTROUSI, 2016); software crowdsourcing reliability (ALELYANI; YANG, 2016); software development communities (ONOUE et al., 2016); streaming software analytics (GOUSIOS et al., 2016); context-based analytics (CITO et al., 2017); code defects (KRISHNA, 2017); development screencasts (ELLMANN et al., 2017); timezone dispersion of project teams (DEVANBU et al., 2017); data-driven systems (THEODOROU et al., 2017); analysis for requirements discovery (MORALES-RAMIREZ et al., 2018); API evolution (BAGHERZADEH et al., 2018); continuous integration (RAHMAN et al., 2018) (BALTES et al., 2018); predicting failures in agile development (BATARSEH; GONZALEZ, 2018); requirement evaluation (HEMMATI et al., 2018); source code analysis and manipulation (COSENTINO et al., 2018); release cycle time patterns (NAYEBI et al., 2019).

Predictive Modeling (N=29)
Issues: prediction model building (MENZIES, 2012) (KOCAGUNELI et al., 2013a) (MINKU et al., 2016) (DAM et al., 2016) (MENZIES, 2018) (DAM et al., 2018); software testing process (TAIPALE et al., 2013); defect and fault prediction (CATAL, 2012) (MISIRLI et al., 2013) (SOLTANIFAR et al., 2016) (JIARPAKDEE et al., 2018) (TANTITHAMTHAVORN; HASSAN, 2018) (HUANG et al., 2018a) (AGRAWAL et al., 2018) (BARBOUR et al., 2018); effort estimation (DEHGHAN et al., 2016) (CHOETKIERTIKUL et al., 2018) (KARNA et al., 2019); software cost estimation (KOCAGUNELI et al., 2013b); predictive modeling performance (KRISHNA; MENZIES, 2018) (KONDO et al., 2019); risk management (CHOETKIERTIKUL et al., 2015); reliability of web-based service (ZHU et al., 2017); software verification (CZECH et al., 2017); change impact analysis (NAYEBI et al., 2017); issue reports (HUANG et al., 2018b); crash-inducing commits analysis (AN et al., 2018); data privacy (PETERS, 2018); issues in software

analytics (MENZIES; ZIMMERMANN, 2018).

Data Mining Method & Tools (N=26)

Issues: open source MDE tools and language (WILLIAMS et al., 2014); plagiarism (ROBLES; GONZÁLEZ-BARAHONA, 2013); data miners (MENZIES, 2013); ranking for bug reports (YE et al., 2014); data stream mining (FINLAY et al., 2014); diagnosis of crashing faults (WU et al., 2014) (WU, 2014) (WU et al., 2018); software documentation (MOSER et al., 2015); software project survival (LOW et al., 2015); traceability of requirements and bugs (HINDLE et al., 2015); mining unstructured data (CERULO et al., 2015); users' feedback (LICORISH et al., 2015); build technology (MCINTOSH et al., 2015); code coverage estimation (ANICHE et al., 2015); conflicts in merging (YUZUKI et al., 2015); GitHub mining (BAYATI et al., 2015); users' feedback collection framework (SUONSYRJÄ et al., 2016); inspections reviews (SINGH et al., 2017); critical changes tracking (TAMLA et al., 2017); search based software engineering (FU; MENZIES, 2017); bug fixing process (MEZOUAR et al., 2018); developers' interaction data (BAO et al., 2018); software data mining tool (DUEÑAS et al., 2018); issue report prioritization of mobile application (NOEI et al., 2019); bugs in software ecosystems (AN et al., 2019).

SA Monitoring (measure, metrics and indicators) (N=15)

Issues: project management (BALDASSARI, 2012); useful software analytics (JOHNSON, 2013); monitoring java application (SUONSYRJÄ; MIKKONEN, 2015); base rates for software analytics (BRUNTINK, 2015); development activity (TREUDE et al., 2015); developer targeted analytics (CITO, 2016); continuous monitoring (BARIK et al., 2016); test and defect coverage (SYED-MOHAMAD et al., 2017); software quality from continuous experimentation (NERI; TRAVASSOS, 2018); software metrics for continuous delivery (HUIJGENS et al., 2017); continuous quality monitoring (JANES et al., 2017); on-time delivery indicator (MANZANO et al., 2018); software quality models (MARTÍNEZ-FERNÁNDEZ et al., 2018); metrics for defect prediction (FALESSI; MOEDE, 2018); dependency network (DECAN et al., 2019).

SA Issues & Concepts (N=14)

Issues: software analytics viability (ROBBES et al., 2013); data-centric decision-making (BAYSAL, 2013); data science activity (KIM et al., 2018) (BEGEL; ZIMMERMANN, 2014); software artifact provenance (GODFREY, 2015); software quality assurance (FOIDL; FELDERER, 2016); software quality models (FELDERER, 2016); role of data scientists (KIM et al., 2016); code-clone detection and analysis (DANG et al., 2017); similarity of software documentation (ELLMANN, 2017); maintenance of critical software (PORT; TABER, 2018); cloud-based applications (NGUYEN et al., 2018); software analytics value (CONBOY et al., 2018); big data in software engineering (ARNDT, 2018).

Visual Software Analytics (N=13)

Issues: agile software process improvement (LEHTONEN et al., 2013); performance monitoring (MUSSON et al., 2013); qualitative analytics (BAYSAL et al., 2013a); software maintenance and evolution (GONZÁLEZ-TORRES et al., 2013a) (TORRES et al., 2013) (GONZÁLEZ-TORRES et al., 2016); test and code measurements visualization (FELDT et al., 2013); requirements management (REDDIVARI et al., 2014); test and defect coverage (HARON; SYED-MOHAMAD, 2015); continuous delivery (MATTILA et al., 2015); project management (MATTILA et al., 2017); concern-driven requirements (NOORWALI, 2018); source code analysis (GONZÁLEZ-TORRES et al., 2018).

SA Platform (N=8)

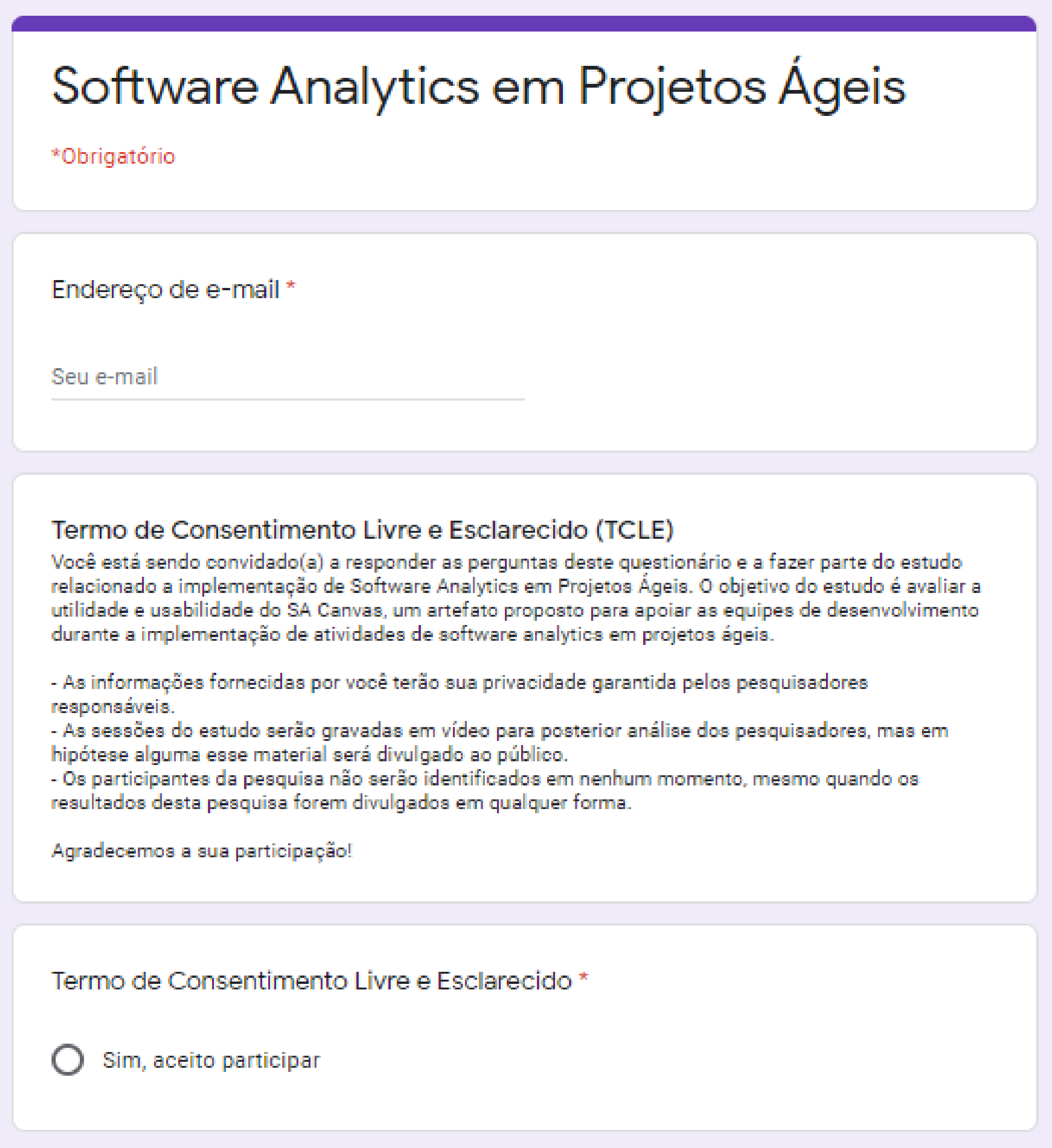
Issues: bug fixer recommendation (SUREKA et al., 2015); code quality (CZERWONKA et al., 2013); maintenance of mobile applications (MINELLI; LANZA, 2013b); repository mining studies (TRAUTSCH et al., 2016) (TRAUTSCH et al., 2018); continuous improvement (LIECHTI et al., 2017a); test analytics platform (LIECHTI et al., 2017b); run-time error feedback (VARGAS et al., 2018).

SA Projects Implementation (N=4)

Issues: software analytics in practice (ZHANG et al., 2013); continuous delivery (HUIJGENS et al., 2018); analytics on team metrics (AUGUSTINE et al., 2018); agile methods and DevOps (SNYDER; CURTIS, 2018).

APPENDIX C - CONSENT FORM

Figure C.1 - Consent form.



Software Analytics em Projetos Ágeis

*Obrigatório

Endereço de e-mail *

Seu e-mail

Termo de Consentimento Livre e Esclarecido (TCLE)

Você está sendo convidado(a) a responder as perguntas deste questionário e a fazer parte do estudo relacionado a implementação de Software Analytics em Projetos Ágeis. O objetivo do estudo é avaliar a utilidade e usabilidade do SA Canvas, um artefato proposto para apoiar as equipes de desenvolvimento durante a implementação de atividades de software analytics em projetos ágeis.

- As informações fornecidas por você terão sua privacidade garantida pelos pesquisadores responsáveis.
- As sessões do estudo serão gravadas em vídeo para posterior análise dos pesquisadores, mas em hipótese alguma esse material será divulgado ao público.
- Os participantes da pesquisa não serão identificados em nenhum momento, mesmo quando os resultados desta pesquisa forem divulgados em qualquer forma.

Agradecemos a sua participação!

Termo de Consentimento Livre e Esclarecido *

○ Sim, aceito participar

SOURCE: Prepared by the author.

APPENDIX D - CANVAS TUTORIAL

Table D.1 presents recommendations for the use of SA Canvas with a brief explanation of each block and the guiding questions to drive users when filling in the canvas components.

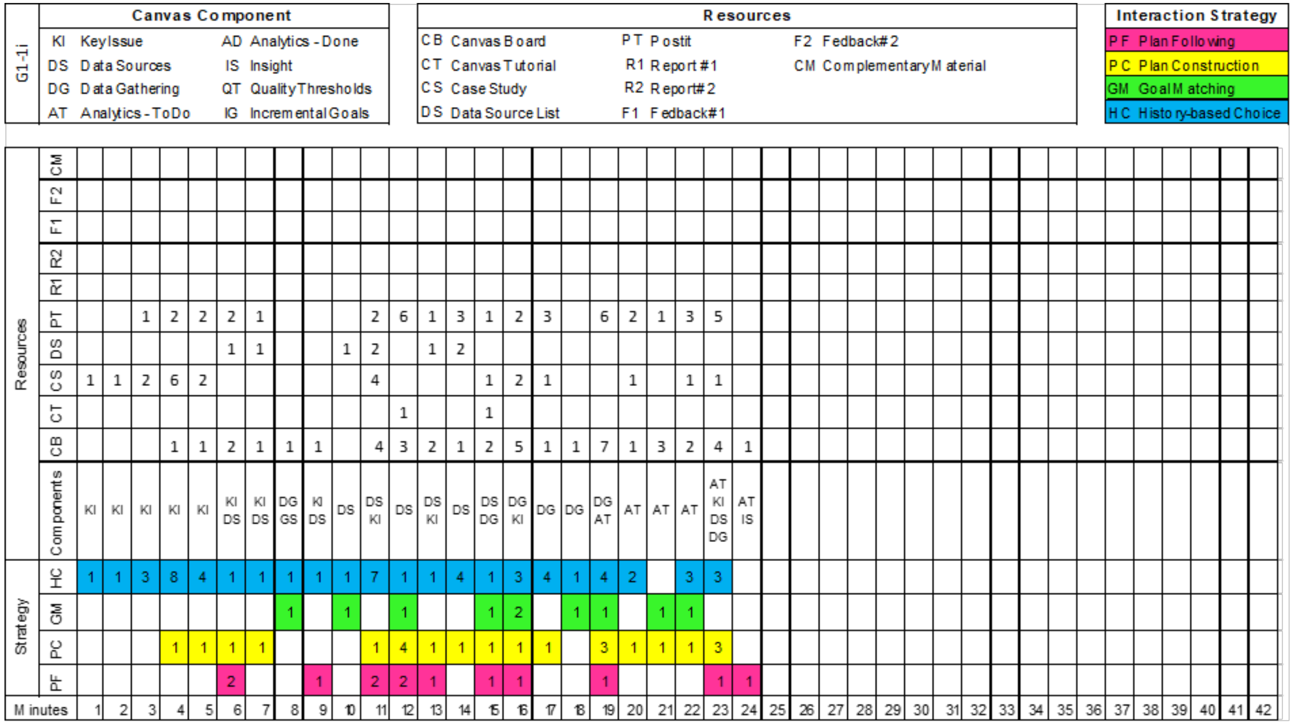
Table D.1 - Recommendations for use of SA Canvas.

#	Element	Description
1	Key Issue	*What does the team want to know?* Identify problems that need to be verified, analyzed and improved - for example, internal / external system quality, productivity or usage patterns.
2	Data Sources	*What data sources can provide information on the issues raised?* Identify the sources from which data and metrics can be extracted to investigate the issues raised.
3	Data Gathering	*How will the data be collected and analyzed?* Define ways to collect and analyze data, such as methods, metrics, scripts, tools, etc.
4	Insights	*What insights emerged after analysis of the results?* Analyze the results obtained from the collected data, discuss possible solutions for decision making, and identify the main insights to guide improvement actions.
5	Quality Thresholds	*What are the acceptable values for maintaining quality standards?* Establish the threshold values to ensure minimum acceptable quality and monitor continuously. Adjust the values when it is possible to achieve higher quality standards.
6	Analytics Implementation	*How will software analytics activities be implemented along with other tasks?* Plan how the software analytics activities will be carried out alongside the development tasks. Use the space to manage the tasks to do (TO DO) and the tasks done (DONE).
7	Incremental Goals	*What are the possible improvement actions to be implemented?* Set achievable goals considering that improvements can be made incrementally. Use this session to manage the tasks to do (TO DO) and the tasks done (DONE).

SOURCE: Prepared by the author.

APPENDIX E - TIMELINES

Figure E.1 - First iteration - Group 1.



G1-1i

Canvas Component			
KI	Key Issue	AD	Analytics - Done
DS	Data Sources	IS	Insight
DG	Data Gathering	QT	Quality Thresholds
AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report#1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																																										
	F2																																										
	F1																																										
	R2																																										
	R1																																										
	PT			1	2	2	2	1				2	6	1	3	1	2	3		6	2	1	3	5																			
	DS						1	1			1	2		1	2																												
	CS	1	1	2	6	2						4				1	2	1			1		1	1																			
	CT												1			1																											
	CB				1	1	2	1	1	1		4	3	2	1	2	5	1	1	7	1	3	2	4	1																		
	Components	KI	KI	KI	KI	KI	KI DS	KI DS	DG GS	KI DS	DS	DS KI	DS	DS KI	DS	DS DG	DG KI	DG	DG	DG AT	AT	AT	AT	AT KI DS DG	AT IS																		
Strategy	HC	1	1	3	8	4	1	1	1	1	1	7	1	1	4	1	3	4	1	4	2		3	3																			
	GM								1		1		1			1	2		1	1		1	1																				
	PC				1	1	1	1				1	4	1	1	1	1	1		3	1	1	1	3																			
	PF						2			1		2	2	1		1	1			1				1	1																		
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

Figure E.2 - Second iteration - Group 1.

G1-2i	Canvas Component		
KI	Key Issue	AD	Analytics - Done
DS	Data Sources	IS	Insight
DG	Data Gathering	QT	Quality Thresholds
AT	Analytics - ToDo	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report #1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Resources	CM																					
	F2																					
	F1	1	2		2	1				1	2											
	R2																					
	R1							3	1	1		2		1	1		1			1	2	2
	PT			2			2	1		1									1	2		
	DS									1												
	CS	1			1	1						1		1		1	1	1				
	CT		1																1			
	CB		1	1			2	3		5	1	1	1	1	1				3	4	3	1
	Components	DG	DG AT	DG	DG	DG	DG	AT AD	DG	DG AT AD	DG	DG AD	KI DS DG AT	DG AT	DG AT	DG	DG	DG	DG AT AD IS	AT AD DG	AT AD	IS KI DS DG
Strategy	HC	1			1	1			1	2	2	2		1		1	2	1	1	2		
	GM		1																1		1	
	PC	1		1	1		1	4		1		1		1	1				2	1	1	1
	PF		1	1			2	1		2	1		1							2	1	1
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
Resources	CM																					
	F2																					
	F1																					
	R2																					
	R1			4		3		2	1		1	1	1	2	1							
	PT		3	1	2	4		1		1	2	1		1	3							
	DS																					
	CS									2												
	CT				1																	
	CB	1	1	3	2	5	2	3			2	1	1	1	3							
	Components	DG	DG	AT IS KI DS DG	IS AT	KI DS DG AT AD IS	IS AT AD	DG AT	AT	AT	AT	AT KI DS DG AD IS	KI DS DG AT AD IS	AT	AT							
Strategy	HC	1	2	1	2	4		1	1	3	1			2	1							
	GM				1			2			1			1	1							
	PC		1	3	1	3					1	2			2							
	PF			1		2	2	1				1	1									
Minutes		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

Figure E.3 - Third iteration - Group 1.

G1-3i	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report#1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																					
	F2	2	1	1																		
	F1																					
	R2			2	3		1	1	1	2	1	2		1	1	1	1	1		1	2	1
	R1																					
	PT				1		2											2				
	DS																					
	CS													1								
	CT			1		1																
	CB	1	1	1	3	4	1	1	1	1	1	6	1		1	1		1	1	1	2	1
	Components	AT IG	AT IG	AT IG	AT AD	QT IG	QT	AD	AD	AD	AD	AT AD IG	IG	IG	IS	IS	AT IG	IG	KI DS DG IG	AD	IG QT	IG QT
Strategy	HC			1	1	1	1	1	1	1	1	1	1	2				2			2	1
	GM			2		2	1					2					1		1			1
	PC	1		1	2		1	1		1		3			1	1		1		1		
	PF	1	1		1	2																
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Resources	CM																					
	F2																					
	F1																					
	R2		2	1	1		1	2		1	1			1	1							
	R1																					
	PT	4		2		1	1		1	1		3	2	3								
	DS																					
	CS																					
	CT																					
	CB	2	2	3	1	1	4	1	1	3	1	2	2	2	1	1						
	Components	IG QT	IG DG AT	IG QT DG	DG AT	AT	AT DG	AT	AT DG	AT AD QT	QT	QT	QT	QT	QT	KI DS DG AT AD IG IS QT						
Strategy	HC	2	1	1	1	1	2	2	1			2		2								
	GM		1													1						
	PC	2		2			1			2	1		1									
	PF		1	1		1	1		1	1		1	1	2	1							
Minutes		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

Figure E.4 - First iteration - Group 2.

G2-1i	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report#1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																																												
	F2																																												
	F1																																												
	R2																																												
	R1																																												
	PT					5	1				1	1	3				1	2	2	2		1	2	2	1	2		1	1	2	1	1	3	1	1	2	1		1	1	2	1			2
	DS																																												
	CS	1	1	1	2	2		1	2	1	1	2		1	1		1	2		1															3		1		2			2	1	2	
	CT								1				1						1		1	1	1				1											1							
	CB					3	1	2			4		3	1	1	1		3	1	2	2	2	4	4	3	3	1	2	2	3	3	2	2	1	2	1	1	1	1	1	1	2	2	1	1
	Components	KI	KI	KI	KI	KI	KI	KI DS DG	KI DS	KI DS	KI	KI DS	KI DS	KI DS	DS	DS	DS	DS DG	DS	DS	KI DS DG	DG	KI DS DG IS QT	DS DG	DG	DS DG	AT	AT	AT	AT DG	AT DG	AT	AT	AT	KI DS DG	AT IS QT	AT DS	DS DG	DG	DG AT	DS DG	AT DG	DS DG	DS	DS
Strategy	HC	1	1	1	3	4		2	2	1	4	3	2	1	1	1	2	4	1	2		3	2	2	1	1		3	1	2	1		3		4	1	2	2	2		1	1	2	2	1
	GM								1			1						1	1		2	1	1	1	1	1	2				1	1						1			1				
	PC					1	1				1		2					1	1	1			1	1	1	1			1	1		1	1	1			1			1		1	1		1
	PF					1		1						1							1		2	1		1				1	2					1			1			1			
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

SOURCE: Prepared by the author.

Figure E.5 - Second iteration - Group 2.

G2-2i	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - ToDo	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback #2
CT	Canvas Tutorial	R1	Report #1	CM	Complementary Material
CS	Case Study	R2	Report #2		
DS	Data Source List	F1	Fedback #1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																																																	
	F2																																																	
	F1	2	1			1				1	1								2	1	1			1																										
	R2																																																	
	R1	1					2			1		1	1		2	1	1				1			1	2	1	1	1	1		1		1	2				1	1	1		1	1	1		1	1		2	
	PT										2	1									1	1	3	1	1			1	1	1		2	1								1	1		1	1		1	1		
	DS							1	1					1									1	1	1					1	1		1			1														
	CS			1	1	1																																												
	CT																															1					1													
	CB	1						1	1		2	1	2	1				1	1	1		2		3	3	3	1		3	1	1	2	2	1	1		1					1	1		1	1		1	1	1
	Components	KI DS DG	KI DS DG	KI DS DG	KI DS DG	KI DS DG	DS AT	DS	DS	KI AD	KI	KI	KI	DS	DS DG	DS	KI DG	KI	KI AT IG	AT IG	KI AT	KI	KI DS DG	DS AT IG	DS AT AD	DS	DS	DS AT	AT AD	DS AT	DS	DS DG	DS DG	KI DS DG AT	DS	DS DG	DS	AT	AT	AT	DG AT	AT	AT IG	AT	AT	AT	AT	AT	AT	KI DS DG AT AD
Strategy	HC	1		1	1	2	1	1	1		3	1	3	1			2	1	2	1	2	1	2	3	1	1		1		2	1	1	1	1				1	1	1	1	1		2		2	1			
	GM																								1							1	1	1			1												1	
	PC	1					1			1		1			2	1							1		2		1	1	1		1	1		1								1			1		1	1		
	PF		1							1			1		1						1		2	1				2				1		1	1	1	1						2						1	1
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49

SOURCE: Prepared by the author.

Figure E.6 - Third iteration - Group 2.

G2-3i	Canvas Component		
KI	Key Issue	AD	Analytics - Done
DS	Data Sources	IS	Insight
DG	Data Gathering	QT	Quality Thresholds
AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback #2
CT	Canvas Tutorial	R1	Report #1	CM	Complementary Material
CS	Case Study	R2	Report #2		
DS	Data Source List	F1	Fedback #1		

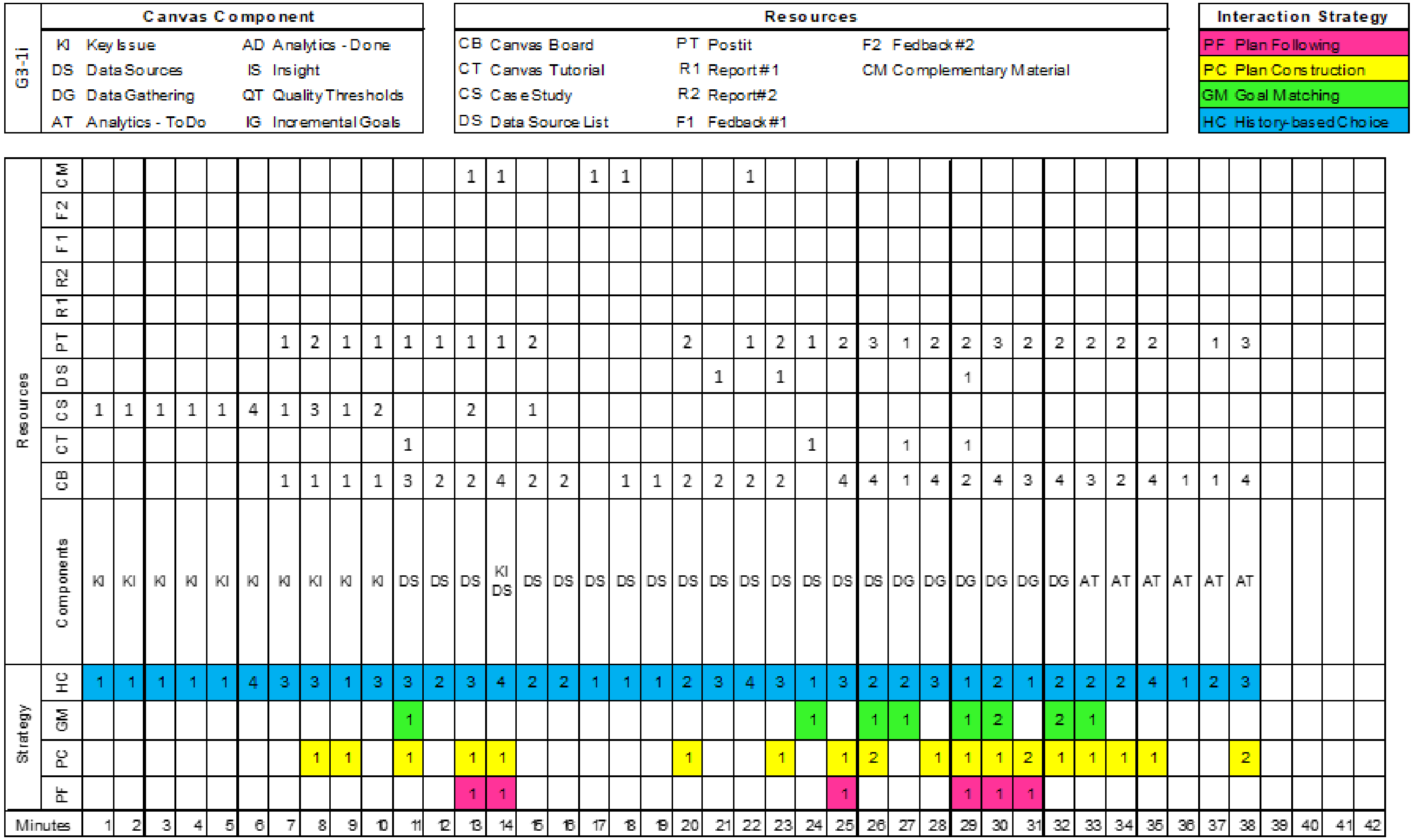
Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																					
	F2																					
	F1																					
	R2	1	1	6	4	1	1	1	1		2		2	1	1	2	1	1	1		1	1
	R1																					
	PT			3	2										2		1	1		1		1
	DS																					
	CS																					
	CT						1			2	1					1	1					
	CB	1		5	4	1		1		1	2	1	1	1	2	1	1	2	1	2		1
	Components	AT AD	AT IS	AT AD	AT AD	AD	IS	IS	IS	IS QT	IG IS	IG IS	IG IS QT	IS	IG IS	IG IS	IG IS	IG QT	IG IS	IS	IG	IG QT
Strategy	HC		1	1				1		1	1			1	2	1	1	1				2
	GM						1			2	2	1				1	2		1		1	
	PC	1			5	4	2		1				1		1	1		1		1		
	PF												1							1		
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Resources	CM																					
	F2																					
	F1																					
	R2	1				1	1	1	1	1		2	1	1	2	1	1			1	2	
	R1																					
	PT		1	1		2				1	1							2				2
	DS																					
	CS																	1				
	CT																	1			1	
	CB	1	1	2	2	1	1	1			1	2	1		1	1	1	2		1	2	2
	Components	QT	IG QT	IG	QT	IS QT	IS	IS	IS	IS	IS	IG	IG	IG	IG	IG	IG	IG	IG	AT IG	AT IG QT	QT
Strategy	HC	1	1		1	2	1	1		2	1	2		1	2	1	1	2		2	1	1
	GM		1		1							1							1		1	1
	PC			1		1			1		1							1			1	1
	PF																					
Minutes		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

Figure E.7 - First iteration - Group 3.



G3-1I	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback #2
CT	Canvas Tutorial	R1	Report #1	CM	Complementary Material
CS	Case Study	R2	Report #2		
DS	Data Source List	F1	Fedback #1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Resources	CM													1	1
	F2														
	F1														
	R2														
	R1														
	PT							1	2	1	1	1	1	1	1
	DS														
	CS	1	1	1	1	1	4	1	3	1	2			2	
	CT											1			
	CB							1	1	1	1	3	2	2	4
	Components	KI	KI	KI	KI	KI	KI	KI	KI	KI	KI	DS	DS	DS	KI DS
Strategy	HC	1	1	1	1	1	4	3	3	1	3	3	2	3	4
	GM											1			
	PC								1	1		1		1	1
	PF													1	1
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14

		15	16	17	18	19	20	21	22	23	24	25	26	27	28
Resources	CM			1	1				1						
	F2														
	F1														
	R2														
	R1														
	PT	2					2		1	2	1	2	3	1	2
	DS							1		1					
	CS	1													
	CT										1			1	
	CB	2	2		1	1	2	2	2	2		4	4	1	4
	Components	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DS	DG	DG
Strategy	HC	2	2	1	1	1	2	3	4	3	1	3	2	2	3
	GM										1		1	1	
	PC						1			1		1	2		1
	PF											1			
Minutes		15	16	17	18	19	20	21	22	23	24	25	26	27	28

		29	30	31	32	33	34	35	36	37	38	39	40	41	42
Resources	CM														
	F2														
	F1														
	R2														
	R1														
	PT	2	3	2	2	2	2	2		1	3				
	DS	1													
	CS														
	CT	1													
	CB	2	4	3	4	3	2	4	1	1	4				
	Components	DG	DG	DG	DG	AT	AT	AT	AT	AT	AT				
Strategy	HC	1	2	1	2	2	2	4	1	2	3				
	GM	1	2		2	1									
	PC	1	1	2	1	1	1	1			2				
	PF	1	1	1											
Minutes		29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

Figure E.8 - Second iteration - Group 3.

G3-2i	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report#1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																				
	F2																				
	F1	2	1	1																	
	R2																				
	R1				1	1	1	1	3	3	1	2	3	1	1		1	1			1
	PT			1					2	2	1				1	3	2	2	2	1	1
	DS											1									
	CS	1																			
	CT											1									
	CB	1		1				1	3	3	1	2	2	3	1	3	2	2	1	1	1
	Components	DS	AD	AD	AD	AD	AD	AD	AT AD IS	AT AD	AT AD	DS IS	DS AT IS	KI	IS	DS AT IS	IS	IS	IS	IS	AT
Strategy	HC												2	1	2	1	1		1		1
	GM											1									1
	PC	1			1			1	3	3	1	1		1		2		1	1	1	
	PF	1		1								1	1	1		1	1	1			
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Resources	CM						1														
	F2																				
	F1																				
	R2																				
	R1	1	1	2	1	2	1			1	2	1			1	2	2		1	1	1
	PT	2		1	1			1	1			2	1		2	2		2			1
	DS																				
	CS																				
	CT																1				
	CB	1	1	1	2	1	1	1	1	1	1		1	2	2	2		1	2	1	2
	Components	AT IS	AT	IS	KI IS	AT	AT	AT	AT	DS AT	DS AT	AT	AT	KI DG AT IS	AT	IS	IS	IS	AT IS	AT IS	KI DS AT
Strategy	HC	1		1		2	1	1				2			2	1		1		1	2
	GM						1				1			1			1		1		
	PC	1	1	2	1				1		1		1		1	2	1	1			
	PF				1			1		1				1		1			1		1
Minutes		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

Resources	CM																			
	F2																			
	F1																			
	R2																			
	R1	1		1	1	1	1		1	1		1	1		1	1	2	1	1	
	PT	1	1	2	2			2	1			1	1	1					1	1
	DS																			
	CS								1				1		1	1				
	CT													1			1			
	CB	1	1		2			1	3	2	3	1		2	1	2	1	1	1	1
	Components	KI AT	AT	AT	AT	AT	AT	AT	KI AT	IG	IG IS	IG	IG	IG	IG	KI IG	AT	AT	AT	AT
Strategy	HC	1		1		1		1	1	1	2	2	2		1	1		1	2	
	GM	1								1				1			2			
	PC		1		1			1	1					1		1	1			1
	PF	1		1	1				1		1						1			
Minutes		41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

SOURCE: Prepared by the author.

Figure E.9 - Third iteration - Group 3.

G3-3i	Canvas Component			
	KI	Key Issue	AD	Analytics - Done
	DS	Data Sources	IS	Insight
	DG	Data Gathering	QT	Quality Thresholds
	AT	Analytics - To Do	IG	Incremental Goals

Resources					
CB	Canvas Board	PT	Postit	F2	Fedback#2
CT	Canvas Tutorial	R1	Report#1	CM	Complementary Material
CS	Case Study	R2	Report#2		
DS	Data Source List	F1	Fedback#1		

Interaction Strategy	
PF	Plan Following
PC	Plan Construction
GM	Goal Matching
HC	History-based Choice

Resources	CM																					
	F2	2	1				2		1		1	2					1					
	F1																					
	R2			1	1	1				2	1	3	1	1		1	2	2	3		2	1
	R1														1	1						
	PT			1					1	1	1			1	1			1		2	2	1
	DS																					
	CS																					
	CT																					
	CB			1			1	1	1	1	2	2	1	2	1	2	3	2	1	1	3	2
	Components	AT IS IG	AT IS	AT AD	AT AD	AT AD	IS	IS	IS	AT IS	AT IS	AD IS	AD	AT AD	AD IS	AT IS	IG	AT IG	AT	AT	AT IS	AT AD
Strategy	HC						2	1	1	1		1			2		2	2		1	2	
	GM									1								1	1			
	PC	1		2	1	1				1	2	2	1	2		1	1		1	1	1	2
	PF	1	1													1			1		1	
Minutes		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Resources	CM																					
	F2																					
	F1																					
	R2	1	2	1	2	1	1	2	1	1		1	1				1					
	R1						1	2			1											
	PT		1	1	2				1		1			1	2		1					
	DS																					
	CS																					
	CT																					
	CB	1	2	1	2	1	1	2	2		1	1	2	2	5	1	2					
	Components	AD	AT AD IS	IG	KI DS DG IS	QT	IG	IG	IG IS	IG	AT IG	IG	AD QT	AT AD QT	KI DS DG AT QT	AT DG	AT IS					
Strategy	HC			2	1	1		2	1		1	1	1		1							
	GM						1							1								
	PC	1	3		1						1		1	1	1		1					
	PF				1				1	1	1				3	1	1					
Minutes		22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42

SOURCE: Prepared by the author.

APPENDIX F - SUGGESTIONS FOR CANVAS DESIGN ENHANCEMENT

This section presents the participant's suggestions to possible improvements in the first version of SA Canvas at the session of participatory design. Participants were asked whether they suggested any changes to the canvas (e.g., rename, move, or take away some component, creating a new component, merging one component with one another), and how it could be improved. Below, we present the responses of the participants for each of the components of the SA canvas.

Component: Key Issue
[P3] - Change name to "Troubleshooting"
[P3] - Change name to "Issues"
[P5] - Change icon to "malicious buy"
[P5] - Change the guiding question to "What does the team want to know about problem software?"
[P3] - Change the guiding question to "What problems should be considered?"
[P1] - Split this block into priorities or sectors.

Component: Data Sources
[P6] - Clarify about the types of data sources in the component description, for example: "database"' vs. some indicator as "response time"

Component: Data Gathering
[P5] - Add a "magnifying glass" to the icon.
[P5] - Change name to " Gathering & Analysis ".
[P5] - Divide the block into " collection " and " analysis ", as they are often distinct activities.

Component: Analytics Implementation
[P5] - Change name to "Active Troubleshoting"
[P5] - Change icon to "gear and key"
[P5] - Change the guiding question to "What activities should be developed to debug problems?"

Component: Insights
[P4] - Change the guiding question to "What information was extracted from the data analysis?"

[P5] - Change the guiding question to “What information was extracted from the previous analysis?”
[P2] - Change the guiding question to “What decision-making (or actions) should be implemented from the information obtained?”

Component: Incremental Goals
[P5] - Change name to “Quality Increment”
[P1] - Change icon to “add” or “insert”
[P5] - Change icon to “gear” with “magnifying glass” and “key”
[P5] - Change the guiding question to “What are the improvement actions for the issue?”

Component: Quality Thresholds
[P5] - Change name to “Minimum Quality Parameters ”
[P1] - Replace the word “thresholds ” with “ standards ”
[P4] - Change the guiding question to “ What are the values for maintaining acceptable quality standards? ”