ASSESSING THE SUCCESS OF AGILE SOFTWARE PROJECTS USING MACHINE LEARNING TECHNIQUES: A CASE STUDY

GÜLHAN KARS

BOĞAZİÇİ UNIVERSITY

2025

ASSESSING THE SUCCESS OF AGILE SOFTWARE PROJECTS USING MACHINE LEARNING TECHNIQUES: A CASE STUDY

Thesis submitted to the

Institute for Graduate Studies in Social Sciences

in partial fulfillment of the requirements for the degree of

Master of Arts

in

Management Information Systems

by

Gülhan Kars

Boğaziçi University

2025

Assessing the Succes of Agile Software Projects

using Machine Learning Techniques: A Case Study

The thesis of Gülhan Kars

has been approved by:

Assist. Prof. Aysun Bozanta Hakyemez _____
(Thesis Advisor)

Prof. Bilgin Metin _____

Prof. Birgül Kutlu Bayraktar _____
(External Member)

January 2025

DECLARATION OF ORIGINALITY

I, Gülhan Kars, certify that

- I am the sole author of this thesis and that I have fully acknowledged and documented in my thesis all sources of ideas and words, including digital resources, which have been produced or published by another person or institution;
- this thesis contains no material that has been submitted or accepted for a degree or diploma in any other educational institution;
- this is a true copy of the thesis approved by my advisor and thesis committee at Boğaziçi University, including final revisions required by them.

Signature.....

Date.....

ABSTRACT

Assessing the Success of Agile Software Projects Using Machine Learning Techniques: A Case Study

Agile methodologies, particularly Scrum, enable software development teams to manage projects more effectively by offering flexibility, collaboration, and small, functional deliveries. However, the lack of systematic frameworks addressing the factors influencing sprint success prevents teams from receiving clear guidance to improve their performance.

This thesis aims to evaluate sprint success in Agile software development processes and provide recommendations to enhance these processes. In this study, factors affecting sprint success were examined using industrial projects from Siemens and a publicly available dataset. Machine learning models were employed to enhance the accuracy of sprint planning and to support teams' decision-making processes with data-driven predictions. Additionally, explainability techniques such as SHAP and LIME were applied to increase the transparency of machine learning models, often referred to as "black-box" systems.

The findings of the research identify key elements contributing to sprint success and provide actionable insights to help teams improve their processes. Moreover, the results demonstrate that data-driven approaches offer more reliable outcomes compared to traditional subjective estimation methods, helping teams achieve sprint goals more effectively. Finally, it is concluded that explainable machine learning models enable software development teams to understand better and trust prediction results, fostering confidence in the use of these tools.

ÖZET

Makine Öğrenme Tekniklerini Kullanarak Çevik Yazılım Projelerinin Başarısının Değerlendirilmesi: Bir Vaka Çalışması

Bu tez, çevik yazılım geliştirme süreçlerinde sprint başarısını değerlendirmek ve bu süreçlerin daha verimli hale getirilmesi için öneriler sunmayı amaçlamaktadır. Çevik metodolojiler, özellikle Scrum, yazılım geliştirme ekiplerine esneklik, iş birliği ve küçük, işlevsel teslimatlar sağlayarak projelerin daha etkin yönetilmesine olanak tanır. Ancak sprint başarısını etkileyen faktörlerin sistematik bir şekilde ele alındığı çerçevelerin eksikliği, ekiplerin performanslarını iyileştirme konusunda net rehberlik alamamasına neden olmaktadır.

Bu tezde, Siemens'ten alınan endüstriyel projeler ve kamuya açık bir veri seti kullanılarak sprint başarılarını etkileyen faktörler incelenmiştir. Makine öğrenmesi modelleri, sprint planlamasının doğruluğunu artırmak ve veri odaklı tahminlerle ekiplerin karar alma süreçlerini geliştirmek amacıyla kullanılmıştır. Ayrıca, "black-box" olarak adlandırılan makine öğrenmesi modellerinin şeffaflığını artırmak için SHAP ve LIME gibi açıklanabilirlik teknikleri uygulanmıştır.

Araştırmanın bulguları, sprint başarısına katkı sağlayan temel unsurları tanımlayarak ekiplerin bu süreçleri iyileştirmesi için somut ve uygulanabilir içgörüler sunmaktadır. Ayrıca, veri odaklı yaklaşımların geleneksel subjektif tahmin yöntemlerine göre daha güvenilir sonuçlar sunduğu ve sprint hedeflerini daha etkili bir şekilde gerçekleştirmeye yardımcı olduğu görülmüştür. Son olarak, açıklanabilir makine öğrenmesi modellerinin, yazılım geliştirme ekiplerinin tahmin sonuçlarını daha iyi anlamalarına ve bu sonuçlara güven duymalarına olanak tanıdığı sonucuna ulaşılmıştır.

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my esteemed thesis advisor, Assist. Prof. Dr. Aysun Bozanta. Her unwavering support, insightful guidance, and exceptional mentorship have been the cornerstone of this thesis. Dr. Bozanta's expertise and dedication to her field have not only shaped the direction of this work but have also profoundly influenced my academic journey. Without her encouragement, constructive feedback, and continuous inspiration, completing this thesis would have been inconceivable. I'm grateful for her belief in my potential and her ability to steer me through challenges, which have been invaluable.

I would also like to thank my friends, who have always been there for me. Their encouragement and support, during both hard times and happy moments, gave me motivation. I am lucky to have friends who stood by me and made this journey easier.

Finally, I would like to thank my family. I am deeply grateful to them for always supporting me, standing behind me in everything I do, and being proud of me no matter what. Their love and encouragement have been my greatest source of strength.

Thank you all for being part of this journey and making this accomplishment possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Agile methodologies have transformed how software is developed, making the process more flexible, collaborative, and focused on delivering value in small steps. Instead of following rigid, traditional plans, Agile allows teams to adapt quickly to changing needs and priorities. This adaptability has become essential in today's fast-paced world. Since the Agile Manifesto was introduced in 2001, Agile practices have become the foundation for modern software development, enabling teams to work more efficiently and collaboratively (Beck et al., 2001).

One of the most popular Agile frameworks, Scrum, provides a simple but effective way to manage work. By organizing projects into short, focused sprints, Scrum helps teams stay aligned on their goals while maintaining the flexibility to adjust plans as needed. Regular reviews and check-ins create opportunities to track progress, identify areas for improvement, and ensure everyone is on the same page (Schwaber & Sutherland, 2020). This approach enhances teamwork and helps deliver quality results, even in dynamic and unpredictable situations.

Despite the many benefits of Agile practices, teams still face challenges. Accurate sprint planning, reliable effort estimation, and consistent performance are often difficult to achieve. Sprint success, a key measure of progress, is sometimes oversimplified as simply meeting the sprint goal. However, true sprint success depends on more than just achieving the goal, it also includes successfully delivering the backlog items planned for the sprint. These items vary in complexity, importance, and interdependencies, making them crucial to overall success (Chow &

Cao, 2008). For Agile teams to deliver value and meet stakeholder expectations, they need a broader understanding of what sprint success really means.

One of the biggest challenges in achieving sprint success is effective planning. Effort estimation, a core part of sprint planning, is often subjective and prone to bias. Methods like Planning Poker rely on personal judgment, which can lead to overly optimistic or inaccurate estimates. This often results in over-committed sprints with incomplete tasks and delays (Meiliana et al., 2023). To improve outcomes, teams need to move away from subjective guesses and adopt more data-driven, objective approaches that help them set realistic goals. Research has shown that structured planning techniques can help teams align backlog items with project goals and use resources more effectively (Ozcelikkan et al., 2022). If we understand the factors that influence sprint success, we can better refine the items, focus on what matters most, improve effort estimation, and ultimately enhance sprint planning. And as a result, this will lead to more effective sprint execution, timely task completion, and greater alignment with stakeholder expectations.

The existing literature lacks a systematic framework for identifying the factors that directly influence sprint success. While many studies explore Agile practices broadly, they do not specifically focus on the measurable elements that impact sprint success. This gap highlights the need for further research to provide Agile teams with actionable insights that can be directly applied to improve outcomes.

To address these gaps, this study focuses on the following research questions:

- RQ1. How effective are machine learning models in predicting sprint success in Agile software projects?

- RQ2. What are the key factors influencing sprint success, and can explainability techniques in machine learning provide actionable insights into these factors?

To answer these questions, this study takes a structured approach to uncover the measurable elements that impact sprint success. First, a comprehensive dataset was compiled from Agile software projects, capturing various factors such as team-specific variables (e.g., team size, experience) and project-specific variables (e.g., task complexity, velocity). Through exploratory data analysis, patterns and correlations were examined to identify key factors influencing sprint outcomes. Following this, machine learning techniques were applied to develop predictive models for sprint success. These models were thoroughly evaluated for their accuracy, precision, and reliability. Furthermore, advanced explainability methods, such as SHAP values, were used to interpret model predictions, offering actionable insights into the most influential factors. This methodology bridges the gap between theoretical understanding and practical application, providing a robust framework for Agile teams to enhance their project planning and execution.

This study makes three key contributions. First, it identifies the critical factors influencing sprint success, providing actionable insights for Agile teams to improve planning and decision-making. These findings are practical and help teams create better strategies, avoid common risks, and achieve their goals more effectively. For example, by understanding how things like task estimation, team speed, and teamwork affect success, teams can better manage their resources and set realistic targets. Second, the study shows how machine learning models can change the way Agile teams plan their work. Instead of relying on guesswork or personal judgment, these models provide clear, data-driven predictions about sprint outcomes. This approach helps teams plan more accurately and make smarter decisions based

on real data. It's a big step forward, making planning more reliable and less stressful for teams. Third, it highlights the importance of explainability in machine learning, ensuring that these models are both transparent and practical for real-world use. If a model gives accurate predictions but people can't figure out how it works, it's not very helpful. This research focuses on making sure the models are clear and simple enough for everyone to use, from project managers to developers. By doing this, the study ensures that these tools are practical and can be trusted in real-world situations. Overall, this research makes advanced machine learning tools more accessible and useful for Agile teams. By combining practical advice, data-driven methods, and easy-to-understand tools, it helps teams improve their sprint success and work more efficiently.

This study is structured as follows: Chapter 2 reviews related work on Agile methodologies, particularly Scrum, and the use of machine learning in software projects. Chapter 3 explains the methodology, including data collection and analysis techniques. Chapter 4 presents the results, followed by a discussion in Chapter 5. Finally, Chapter 6 concludes the study with key findings and suggestions for future research.

CHAPTER 2

LITERATURE REVIEW

This literature review explores three key areas: the role of Agile methodologies, especially Scrum, in software development, the factors influencing sprint success, and the application of machine learning models and explainability techniques in Agile project management.

2.1 The role of Agile methodologies in software development

Agile methodologies have brought major changes to software development by focusing on flexibility, teamwork, and meeting customer needs. Unlike traditional methods like the Waterfall model, which is often criticized for being too rigid and unable to handle changes, Agile approaches are designed to adapt to evolving requirements. Agile practices, inspired by the Agile Manifesto (Beck et al., 2001), emphasize "individuals and interactions over processes and tools" and "responding to change over following a plan." Popular Agile methods, such as Scrum, Kanban, and Extreme Programming (XP), use techniques like daily stand-ups, sprint planning, reviews, and retrospectives to promote collaboration and continuous improvement (Ghimire & Charters, 2022; Kuhrmann et al., 2022).

Adopting Agile methodologies leads to many benefits. For one, Agile encourages close collaboration between developers, customers, and stakeholders to ensure the final product meets user expectations (Drury-Grogan, 2014). Teams can also adjust to changes even late in the development process (Schwaber, 1996), and regular testing helps catch and fix issues early (Kuhrmann et al., 2022). Agile's

approach of delivering small, functional parts of a product incrementally means faster results for users (Ghimire & Charters, 2022).

However, implementing Agile isn't without challenges. Organizations used to traditional, top-down structures may find it hard to adapt to Agile's self-organizing teams (Drury-Grogan, 2014). Agile also demands skilled team members and sufficient resources, which might not always be available. Additionally, some organizations create hybrid models that mix Agile with other approaches, which can dilute Agile's core principles (Kuhrmann et al., 2022).

Studies show that Agile practices can significantly improve project outcomes. Teams using a combination of Agile techniques often communicate better, set clearer priorities, and achieve higher success rates (Ghimire & Charters, 2022). Drury-Grogan (2014) found that aligning team goals with project success factors, such as quality, schedule, and budget, plays a key role in success. At the same time, Schwaber (1996) emphasized the importance of balancing Agile's flexibility with effective management to handle the complexities of software development.

In summary, Agile methodologies have reshaped software development by emphasizing adaptability, collaboration, and delivering value to customers. While challenges remain, the evidence strongly supports Agile as a powerful approach to achieving better project outcomes and driving innovation in the field.

2.2 Factors influencing sprint success

Scrum is a widely used method in agile software development that relies on short, focused work cycles called sprints to deliver results efficiently. Achieving success in Scrum involves good planning, strong teamwork, and ongoing improvement Schwaber & Sutherland, 2020). Scrum focuses on enabling teams to deliver and

release software as frequently as needed to provide value (Hoda & Noble, 2017). What makes Scrum unique is its structure of defined events and roles, which keep the team aligned and the process efficient. Each sprint begins with a sprint planning session, where the team decides what they can accomplish in the sprint and sets a clear goal. During the sprint, the team holds a quick daily Scrum, a 15-minute meeting often called a stand-up, where members share progress, flag challenges, and coordinate their efforts for the day. Once the sprint ends, the team showcases their work in a sprint review, inviting feedback from stakeholders to make sure the product meets expectations. Afterward, they hold a sprint retrospective, where they reflect on what went well, what didn't, and how to improve for the next sprint (Rubin, 2012). Key factors that contribute to sprint success include having experienced and skilled team members, well-defined roles, and effective meetings such as sprint planning, daily stand-ups, and retrospectives. These practices ensure teams stay organized and aligned with their goals (Kadenic, Koumaditis, & Junker-Jensen, 2023).

As described in the 2020 Scrum Guide, "Sprints are the heartbeat of turning ideas into value" (Schwaber & Sutherland, 2020). When teams manage their sprints effectively, they can meet project requirements and achieve the desired outcomes more easily. The studies underscore the importance of using predictive tools and data-driven approaches to foresee potential problems in sprints. By analyzing past performance and identifying risks early, teams can adjust plans to minimize delays and improve outcomes (Choetkiertikul et al., 2017). Another significant contribution is the role of metrics in Agile processes, as highlighted by Kupiainen, Mäntylä, and Itkonen (2015). Metrics like velocity, burndown charts, and effort estimation are

critical for tracking progress, making adjustments, and ensuring efficiency throughout the sprint.

Human factors also play a crucial role in sprint success. Studies show that trust, clear communication, and team cohesion significantly impact performance. Regular feedback during retrospectives enables teams to address issues promptly and adapt effectively (Kortum et al., 2020). Additionally, team maturity, as discussed by Kadenic et al. (2023), influences how well teams can self-organize and adhere to Scrum principles. Support from management and alignment with organizational objectives further enhance the chances of successful sprint delivery (Morandini et al., 2021).

Moreover, Sharma, Kumar, and Fayad (2020) emphasize the importance of balancing time spent on Scrum ceremonies with actual product-building tasks. Efficient time management ensures teams do not compromise productivity while adhering to Scrum practices. Metrics for on-time delivery, as studied by Kula et al. (2022), reveal that factors like task dependencies, team familiarity, and proper requirements refinement are pivotal for meeting sprint deadlines.

Effective sprint planning is another critical component. Golfarelli, Rizzi, and Turricchia (2013) discuss how optimization models can help prioritize tasks and manage resources to maximize business value. Similarly, adaptive strategies like multi-sprint planning and smooth replanning enable teams to respond flexibly to changes without disrupting the overall project (Golfarelli et al., 2013).

In conclusion, sprint success in Scrum is influenced by a combination of technical, organizational, and human factors. Leveraging predictive tools, using relevant metrics, maintaining effective communication, and adhering to Scrum ceremonies are essential for consistent success. By addressing these areas, teams can

optimize their processes and achieve better outcomes in agile software development projects.

2.3 Application of machine learning models and explainability techniques in Agile project management

Machine learning (ML) has become an essential tool in Agile software development, making it easier for teams to handle challenges like effort estimation, sprint planning, and risk management. By analyzing historical project data, ML models uncover patterns, provide accurate predictions, and support teams in making smarter, data-driven decisions. Highlighting how integrating ML into Agile workflows can improve planning and decision-making, especially in handling complex and changing requirements (Ranawana & Karunananda, 2021). Similarly, Abadeer and Sabetzadeh (2021) show that ML can improve the accuracy of effort estimation by analyzing historical backlog data, reducing reliance on subjective judgement and making planning more reliable.

However, using ML in Agile environments is about more than just accuracy, it also requires transparency. Explainability, or the ability to interpret and understand ML models, has emerged as a critical focus area in recent research, particularly in domains requiring high transparency, such as software development. While ML models have demonstrated strong predictive capabilities, their "black-box" nature often limits their adoption in contexts where interpretability is vital for decision-making (Doshi-Velez & Kim, 2017). In agile software development, where adaptability and iterative feedback are central, explainable machine learning can significantly enhance trust and usability by providing actionable insights to project managers and development teams. Explainability techniques, such as SHAP

(SHapley Additive exPlanations) (Lundberg & Lee, 2017) and LIME (Local Interpretable Model-Agnostic Explanations) (Ribeiro et al., 2016), can help expound which factors most significantly influence the model's predictions, making the outcomes easier to trust and apply in Agile workflows.

There has been significant research on developing prediction models to aid software development processes. For instance, many existing effort estimation models, such as those proposed by Mockus et al. (2003), Sarro et al. (2016), and Kocaguneli et al. (2012), focus on predicting the effort required to complete an entire software project rather than estimating effort for individual iterations. Similarly, empirical studies, including those by Huang et al. (2010), Xu et al. (2007), Hu et al. (2013), Fang and Marle (2012), and Moreno-García et al. (2008), have primarily explored prediction approaches at the project level, leaving sprint-specific prediction relatively unexplored. Insights and predictions at the sprint level provide project managers with actionable information, enabling them to proactively address potential delivery risks and make necessary adjustments during the sprint (Choetkiertikul et al., 2018).

Existing research does not provide a clear framework to identify the specific factors that influence sprint success. Many studies focus on Agile practices as a whole but often fail to explore the measurable elements that directly affect sprint outcomes. This lack of focus creates a significant gap, leaving Agile teams without clear guidance on which elements to optimize to enhance their performance. Addressing this gap is crucial, as providing actionable insights and data-driven strategies tailored to sprint execution can empower teams to refine their workflows and achieve more consistent, successful outcomes.

CHAPTER 3

METHODOLOGY

The methods and techniques used to analyze the data and produce the results are explained here. It starts by outlining how the company dataset was collected, public data was found, cleaned, and prepared to ensure they were ready for analysis. How the features that are believed to influence sprint success were determined, and what these features represent, are explained. Next, the algorithms applied for to predict sprint success are introduced. Traditional machine learning methods and more advanced deep learning techniques are chosen to capture both simple and complex patterns in the data. The evaluation metrics used to assess the models' performance are then discussed, focusing on how they help determine the accuracy and reliability of the predictions. Finally, the explainability methods and the reasons for using them are explained. The aim of this section is to provide a clear and explanation of the techniques used, making the process transparent while ensuring the findings are both valid and practical.

3.1 Data collection and preparation

The data used in this study was collected from six ongoing software projects in the field of industrial automation at Siemens. These projects are following Scrum practices according to the Scrum Guide, where sprints are evaluated for success based on planning accuracy and team performance. They run two weeks sprints and use all principles and practices. This includes maintaining a comprehensive backlog containing all planned and unplanned items about the products. During sprint planning, the teams pull items from their backlog and create their sprint backlog. To

manage their Scrum processes, the projects uses Azure Boards[1] which is a tool that provides tracking and managing work items, as well as querying and analyzing historical sprint data and analytics.

For the purpose of the study, each project examined individually first. Historical sprint data was queried using Azure Boards queries, focusing on retrieving the items planned for each sprint. Queries were constructed to extract specific data points related to sprint items, and these outputs are extracted as Excel for further analysis. For these specific data points, it was necessary to first identify the relevant features that can influence sprint success, as these features determined the structure of the queries and the scope of the data extracted. This systematic approach ensured the data was comprehensive and aligned with the objectives of the study.

Scrum suggested that story points should be used to represent the effort, complexity, uncertainty, and risks involving resolving an item (Cohn, 2005). Each item has a story point which is assigned before the sprint starts. The work completed during a sprint is measured as velocity (eq. (1)), which represents the total number of story points completed within that period. Velocity indicates the total amount of work a team accomplishes during a single iteration. Velocity of a set of issues I is the sum story points of all the items in I,

$$\text{velocity(I)} = \sum_{\text{i}\in\text{I}} \text{sp(i)} \tag{1}$$

where $sp(i)$ is the story point assigned to item i.

After selecting the projects, we first defined our dependent variable to analyze sprint success. Sprint success was essentially about whether the team

[1] https://azure.microsoft.com/en-us/products/devops/boards

achieved the goals they set for themselves. This goal represents the team's ability to achieve the velocity they planned for the sprint. To quantify how much of this goal was reached, we defined a metric called “Success Level."

The team estimates the work they can complete at the end of the sprint. This estimation is referred to as planned velocity. On the other hand, actual velocity is the total amount of work the team actually completes by the end of the sprint. Another way to define actual velocity is the total work marked as “done” during the sprint. Using these two metrics, we calculated Success Level as follows:

$$Success\ Level\ = \frac{velocity\ (Actual)}{velocity(\mathrm{P}lanned)} \quad (2)$$

The classification of success into three levels—High, Medium, and Low—was established in collaboration with Agile experts at Siemens, including Scrum Masters, Product Owners, and experienced developers. These experts provided insights based on their practical experience to define thresholds for each level:

- High Success: A sprint where the result is 75% or higher (≥0.75), indicating that the team completed at least three-quarters of their planned work.
- Medium Success: A sprint where the result falls between 50% and 75% ($0.50 \leq$ Success Level < 0.75), reflecting moderate achievement.
- Low Success: A sprint where the result is less than 50% (<0.50), indicating that less than half of the planned work was completed.

This method, validated by domain experts, provides a robust way to evaluate sprint performance and categorize outcomes. By aligning success levels with both quantitative data and professional judgment, it ensures that the framework is both practical and theoretically sound. Furthermore, the categorization helps teams

identify key patterns influencing sprint success and enables them to address areas for improvement systematically.

For feature selection, literature and industry good practices in agile software development are examined. A set of features is identified based on their relevance to sprint success. Studies show that metrics related to quality, quantity and characteristics of sprint items have an influence on sprint outcomes. For instance, research by Cohn (2005) highlights the importance of understanding task complexity, effort estimation, and backlog item quality in agile planning. While determining the effectiveness of sprint planning and execution, Rubin (2012) and Schwaber and Sutherland (2020) say that features like team size, item descriptions, and acceptance criteria play a critical role. Studies in defect management (Fenton & Neil, 1999) also underline the significance of bug-related metrics, such as severity levels and missing information, in predicting project performance. So, with all of this information, a set of features is created as shown in Table 1.

A total of 18 features are identified as potentially influencing sprint success. Some of these are related to the type and number of items included in sprint, while others related with characteristics of the items themselves. For example, the description of an items is the part that explains the details of the item. If an item has no description, then it is included in the percentage of items with no description. To evaluate whether a description is sufficient, the character count in the description is used. To define what makes a description "sufficient," we worked with experts from Siemens, including experienced developers, architects, and test professionals. These experts reviewed items they considered well-defined and agreed that descriptions averaging around 500 characters included enough detail to make the work clear and actionable. Based on their input, descriptions with more than 500 characters were

categorized as sufficient, while those with fewer than 500 characters were considered insufficient. This method reflects the practical experience of Agile teams, ensuring that the evaluation of item descriptions aligns with real-world expectations. By relying on expert insights, this approach provides a clear and reliable way to assess the quality of item descriptions, helping teams focus on improving sprint outcomes.

Table 1. Features of a Sprint for Company Data

Feature	Description
PBI Percentage	Percentage of Product Backlog Items (PBIs) planned in a sprint
Bug Percentage	Percentage of Bug Items planned in a sprint
Number of Items	Total count of items planned in a sprint
No Description Item Per	Percentage of items missing a description
Sufficient Description Item Per	Percentage of items with a sufficient description
No Repro Steps Per (Bug)	Percentage of bug items missing the actions taken to find or reproduce the bug and expected behavior
No System Info Per (Bug)	Percentage of bug items missing the information about the software and system configuration that is relevant to the bug and tests to apply
Acceptance Criteria Per (PBI)	Percentage of PBIs missing the criteria to meet before the bug can be closed.
Sev1-2 Per (Bug)	Percentage of bug items with critical and high impact on the project or software system
Sev 3 Per (Bug)	Percentage of bug items with medium impact on the project or software system
Sev 4 Per (Bug)	Percentage of bug items with low impact on the project or software system items
Effort XS	Percentage of bug items with a story point 1
Effort S	Percentage of bug items with a story point 3
Effort M	Percentage of bug items with a story point 5
Effort L	Percentage of bug items with a story point 8
Effort XL	Percentage of bug items with a story point 13
Effort XXL	Percentage of bug items with a story point 21
Team Size	Number of Scrum team members

To reproduce a bug, developers and testers need certain information about the bug. Two key pieces of information are the Steps to Reproduce (friendly name: Repro Steps) and System Information. Since these two features are specific to bugs,

their values were determined by examining only the bug items. Similarly, severity is also a value specific to bugs. It is defined as "a subjective rating of the impact of a bug or work item on the project or software system" (Microsoft, n.d.).

- 1 - Critical: Must fix. A defect that causes termination of one or more system components or the complete system or causes extensive data corruption. No viable alternative methods are available to achieve the desired results.
- 2 - High: Consider fix. A defect that causes termination of one or more system components or the complete system or causes extensive data corruption. However, there are acceptable alternative methods to achieve the desired outcomes.
- 3 - Medium: A defect that causes the system to produce incorrect, incomplete, or inconsistent results.
- 4 - Low: A minor or cosmetic defect that has acceptable workarounds to achieve required results.

In Scrum, effort refers to the estimated amount of work, complexity, and uncertainty involved in completing a backlog item. It is typically measured in story points, which provide a relative scale to compare the size or difficulty of tasks. Story points account for factors such as the task's complexity, risks, and required effort, enabling teams to estimate work in a way that is independent of time. Effort estimation plays a critical role in Agile software development, as it directly influences the planning and execution of sprints (Meiliana et al., 2023).

Story points are numerical values assigned to user stories to estimate the relative effort required for implementation. Teams often use the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21) for these estimations, as it reflects the increasing uncertainty associated with larger tasks (Scrum.org, n.d.). The percentages of items based on their effort were included as part of the sprint features.

Considering the importance of team characteristics, factors such as team size, motivation, and interpersonal skills are critical for Agile performance (Salido et al., 2023). For this reason, team size was included as one of the sprint features in this study.

After identifying all these features, queries were created to extract these attributes for every sprint in each project. First, these queries listed every item within a sprint. Subsequently, the sprints were consolidated, converting the item-based Excel sheets into sprint-based Excel sheets. Thus, company dataset was constructed.

For the second part of this study, a publicly available dataset was used, sourced from the work by Choetkiertikul et al. (2018), titled "Predicting Delivery Capability in Iterative Software Development." This research focuses on predicting delivery outcomes in iterative, agile software development processes by analyzing past iterations from five large open-source projects. The dataset contains 3,834 iterations and 56,687 associated work items, offering a comprehensive resource for studying sprint performance and delivery capability. Their goal was to predict the difference between planned and delivered velocity within an iteration, which serves as a measure of delivery capability. However, predictions were made at different stages of the sprint (referred to as "prediction times") to evaluate how predictive accuracy changes based on the timing of the prediction. Specifically, predictions were evaluated at the beginning of the sprint and when the sprint had progressed to 30%, 50%, and 80% of its planned duration.

For our study, however, we focused solely on predictions made at the beginning of the sprint, aligning with our objective of identifying factors influencing sprint success before the sprint starts. Therefore, we excluded data derived mid-sprint (at 30%, 50%, and 80% progress) from the public dataset. This ensured that

our analysis was consistent with our goal of predicting sprint success based on the features available during sprint planning. By narrowing the scope to the data at the start of the sprint, we concentrated on understanding pre-sprint factors that can be leveraged for better sprint planning. Moreover, in the original study, velocity-related features were included as factors for measuring delivery capability. However, since sprint success is directly tied to velocity, we do not consider these features suitable for the analysis. Therefore, these were also excluded from the dataset. As a result, we conducted our analysis using 2,572 sprints and 9 sprint-related features (Table 2).

Table 2. Features of a Sprint for Public Data

Feature	Description	Variable name in dataset
Sprint duration	The number of days from the planned completion date	planday
No. of issues at start time	The number of issues assigned to a sprint at the beginning	no_issue_starttime
No. of issues added	The number of issues added during a sprint (between start time and prediction time)	no_issue_added
No. of issues removed	The number of issues removed during a sprint (between start time and prediction time)	no_issue_removed
No. of to-do issues	The number of to-do issues in a sprint by prediction time	no_issuetodo
No. of in-progress issues	The number of in-progress issues in a sprint by prediction time	no_issueinprogress
No. of done issues	The number of done issues in a sprint by prediction time	no_issuedone
Scrum master	The number of Scrum masters	no_admin
Scrum team members	The number of team members working on a sprint	no_teammember

3.2 Exploratory data analysis

Exploratory data analysis (EDA) plays an essential role in identifying patterns, relationships, and irregularities within datasets, offering valuable insights for subsequent analysis and decision-making (De Mast & Kemper, 2009). By performing EDA, potential issues like missing data, imbalanced distribution within

dataset can be found early in the process. It can be ensured that the data is clean and ready for analysis.

First, descriptive statistics were used to summarize the key characteristics of the data, such as the range, mean, and standard deviation for each feature. These descriptive analyses allowed us to examine the diversity in the dataset and identify whether there were any anomalies. To explore relationships between features, a correlation map was generated and analyzed. This visualization highlighted how features were related to each other and to the target variable, sprint success. Understanding these relationships helped identify highly correlated features that could be influential in predicting success levels, as well as those that might lead to redundancy.

The distribution of success levels (Low, Medium, High) was also examined to identify any imbalances in the data. This analysis revealed whether one class was over- or under-represented, providing critical input for deciding whether resampling techniques were necessary to balance the dataset. Additionally, a histogram was created for each feature to visualize their distributions separately. They allow for an intuitive understanding of how data points are spread across various values, making it easy to identify patterns such as normal distributions, skewness, or irregularities.

3.3 Resampling

Since there was an unequal distribution among categories in both the company data and the public data, we decided to apply resampling. SMOTE plays a critical role in addressing class imbalance by generating synthetic samples for the minority class, thus preventing models from being biased toward the majority class and improving classification performance (Chawla et al., 2002). In professional world, it's normal to

see low success sprints more. So, this imbalance can be explained that it’s not typical for teams to experience frequent failures. Teams actively review their shortcomings during retrospectives and make adjustments within or after each sprint to improve performance. As a result, it's expected that "Low Success" sprints are relatively rare, and "Medium Success" sprints occur less frequently compared to "High Success" ones. To address this imbalance, the SMOTE resampling method was applied to both the company and public datasets. SMOTE generated synthetic samples for the minority categories, effectively resolving the imbalance problem. This process equalized the distribution of categories across the datasets, ensuring a fairer and more reliable analysis of sprint success predictors.

3.4 Algorithms

We utilized nine machine learning methods in this study, including traditional machine learning algorithms and deep learning models. These methods were selected based on their popularity in the literature, proven effectiveness across various domains, and ability to handle complex datasets. The selection aimed to ensure a balanced evaluation of both interpretable models, like Logistic Regression and Decision Trees, and more complex algorithms, such as Gradient Boosting and Neural Networks, which can capture intricate patterns in data.

Logistic Regression is one of the most widely used classification algorithms for predicting the probability of an outcome based on input features. It models the relationship between a dependent variable and one or more independent variables using a logistic function. This method is particularly effective for datasets where the target variable is categorical, such as binary or multi-class classifications (Hosmer et al., 2013). Since we have three level of success, multinomial (multi-class) version is

used. It can predict the probability for each class and explain the relationships between input features and the success level categories. Logistic Regression is highly valued for its simplicity, interpretability, and robustness, and ability to handle various types of data.

Decision Trees are intuitive models that split data into subsets based on specific feature values, forming a tree-like structure that is both easy to understand and interpret. They are particularly effective for categorical explanatory variables, and even better when they are binary, as they naturally handle interactions between features without requiring explicit modeling, a strength highlighted by their ability to uncover hidden relationships in data (Breiman, Friedman, Olshen, & Stone, 1984). Moreover, Decision Trees are highly explainable, making them suitable for use in domains where interpretability is crucial, such as healthcare (Kavitha et al., 2021). Decision Trees are simple and effective, but they have some drawbacks. They are often overfit the training data when allowed to grow without limits, which can make them less reliable on new data (Mienye & Jere, 2024). Their reliance on binary splits can also lead to inconsistent results, as the trees are sensitive to changes in the training data. To address these issues and make the model more stable and accurate, methods like pruning or ensemble techniques, such as Random Forest, are often applied (Breiman et al., 1984). Based on this understanding, the next step in this study is to use Random Forest for improved predictions.

Random Forest is a machine learning method that builds multiple decision trees using random subsets of data and features. It combines their predictions, either by majority vote for classification or averaging for regression, to improve accuracy and reduce overfitting. The randomness in selecting features for splits ensures diversity among the trees, which helps lower errors and improve generalization

(Breiman, 2001). Additionally, it highlights the importance of different features, making it both effective and easy to interpret.

Support Vector Machines (SVMs) is a powerful algorithm that finds the optimal hyperplane to separate data points into different classes. This hyperplane is chosen to maximize the margin between the classes, which ensures better generalization to new data. SVMs are particularly effective for both linear and non-linear classification tasks, with the latter handled using kernel functions that transform the data into higher-dimensional spaces where a linear separation is possible (Cortes & Vapnik, 1995). SVM is particularly useful for datasets with high dimensionality or when the number of features exceeds the number of samples.

XGBoost, a scalable and efficient gradient-boosting algorithm, is designed to handle supervised learning tasks. It builds decision trees sequentially, with each new tree correcting the errors of its predecessor. Known for its computational speed and performance, XGBoost has gained widespread popularity in machine learning competitions and real-world applications (Chen & Guestrin, 2016).

Gradient Boosting is a powerful ensemble learning technique that builds models step by step to reduce errors. It works by combining the predictions of simpler models, often decision trees, to create a more accurate and robust overall model. This method is particularly effective for tackling complex datasets and has shown great success in a wide range of applications (Friedman, 2001).

LightGBM is a highly efficient gradient-boosting framework designed to handle large-scale data quickly and effectively. One of its key features is its histogram-based technique for splitting data. Unlike traditional gradient boosting methods, which evaluate all possible splits for continuous features, LightGBM groups feature values into discrete bins, significantly reducing the computational

burden and memory requirements. This innovative approach allows LightGBM to process large datasets more efficiently while maintaining high accuracy. Additionally, its ability to handle complex relationships in data and its scalability make it a good choice for tasks involving extensive and high-dimensional datasets (Ke et al., 2017).

DNNs are a type of artificial neural network with multiple layers between the input and output. These networks are capable of learning complicated interactions between variables that influence the target outcome. However, they often require extensive data and computational resources (LeCun et al., 2015). To further improve the robustness of the DNNs used in this study, Dropout layers were incorporated into their architecture. As explained by Choetkiertikul et al. (2018), Dropout is a regularization technique that temporarily deactivates random neurons during training. This technique ensures that the model doesn't become too dependent on certain neurons, helping it avoid overfitting and making it better at handling new data. DNN models could maintain consistent and high performance, even when faced with diverse and variable datasets, which is particularly important in the context of project management and software development predictions.

FCNNs are a subset of DNNs where every node in one layer is connected to every node in the next layer. This design allows FCNNs to efficiently process and integrate information across all features, making them particularly effective for capturing non-linear relationships in data. Such networks are often used for general-purpose prediction tasks and have demonstrated success in various domains due to their ability to model complex patterns. In project management and iterative software development, FCNNs have shown great promise. For example, Choetkiertikul et al. (2018) highlight the role of neural networks in predicting delivery capabilities. They

emphasize the utility of neural architectures, including FCNNs, for effectively learning from aggregated features of tasks or issues. This ability to generalize and capture intricate dependencies between features ensures robust predictive performance across diverse scenarios.

3.5 Hyperparameter tuning

Hyperparameter tuning is an essential step in building machine learning models, especially for applications like project management analytics, where prediction accuracy significantly influences decision-making. As Simon et al. (2023) point out, machine learning models are often very sensitive to factors like random seeds and how the data is split for training and testing. However, optimizing hyperparameters can make a huge difference, dramatically improving a model's accuracy, efficiency, and ability to generalize. In fact, proper tuning can transform a model from delivering average results to achieving top-tier, state-of-the-art performance, which is especially crucial for effective predictive analysis in project management.

The defined hyperparameter ranges were based on recommendations from the literature and preliminary experiments. This approach ensured that the parameters explored were both practical and tailored to the needs of project management tasks. For instance, the learning rate was adjusted step by step to find the best value, while keeping computational costs manageable.

In this study, Grid Search was selected because it offers a systematic and comprehensive way to explore all possible combinations of predefined hyperparameter values. This thorough approach makes it especially effective for finding the best settings to optimize machine learning models. According to Nevendra and Singh (2022), Grid Search consistently delivers better outcomes

compared to alternative methods like Random Search. Their empirical analysis proved that Grid Search achieved a performance improvement of 4.42%, outperforming Random Search's 3.36%, showing its effectiveness in optimizing hyperparameter-sensitive models. Furthermore, Grid Search provides reproducibility and standardization, making it a reliable choice for applications where model robustness is critical (Nevendra & Singh, 2022).

3.6 Evaluation metrics

Evaluation metrics are key to understanding how well a machine learning model performs. In this study, we used accuracy, precision, recall, and F1-score to evaluate the models, as each metric offers a unique insight into performance.

Accuracy measures how many predictions the model got right out of all the predictions made (Eq. (3)). While it's straightforward and easy to understand, accuracy can be misleading when the dataset is imbalanced. For example, if one class dominates the data, a high accuracy might simply reflect the model's bias toward that majority class (Sokolova & Lapalme, 2009).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3)$$

Where:

- TP: True Positives – correctly predicted positive cases
- TN: True Negatives – correctly predicted negative cases
- FP: False Positives – cases incorrectly predicted as positive
- FN: False Negatives – cases incorrectly predicted as negative

Precision focuses on how many of the positive predictions were actually correct (Eq. (4)). This is important in situations where false positives can be costly

or problematic, such as spam detection. A high precision means the model is good at avoiding unnecessary false alarms (Goutte & Gaussier, 2005).

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

Recall looks at how many of the actual positive cases the model correctly identified (Eq. (5)). This metric is crucial in cases where missing positive instances (false negatives) can have serious consequences, like in medical diagnoses or fraud detection. High recall ensures the model captures as many positives as possible (Sokolova & Lapalme, 2009).

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

F1-Score strikes a balance between precision and recall (Eq. (6)). It combines both into a single number by taking their harmonic mean, making it particularly useful for datasets where classes are imbalanced. It provides a more holistic view of the model's ability to handle both false positives and false negatives (Goutte & Gaussier, 2005).

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (6)$$

These metrics were chosen to give a detailed and balanced understanding of the models' performance, helping ensure that they perform well under different conditions and dataset characteristics.

3.7 Explainability

Explainability in machine learning is crucial for understanding how models make predictions, especially in fields like project management where decisions need to be interpretable and actionable. Transparent models help build trust with stakeholders,

ensure accountability, and make it easier to identify biases or errors in the predictions. Explainability is essential to ensure that model is trustable by uncovering the reasoning behind predictions, it enhances the usability of machine learning models in real-world applications (Ribeiro, Singh, & Guestrin, 2016).

Model-embedded feature importance is a straightforward and efficient way to assess how much each feature contributes to the model's predictions. Algorithms like Random Forest, Gradient Boosting, and LightGBM provide built-in measures of feature importance based on metrics like impurity reduction or split gains. For example, Random Forest evaluates feature importance by averaging the reduction in node impurity across all trees in the ensemble (Breiman, 2001). This approach is not only computationally efficient but also provides an intuitive understanding of which features are most influential, helping users focus on the factors that truly drive model predictions.

Shapley Additive Explanations (SHAP) offers a more advanced and detailed way to interpret model predictions by assigning each feature a contribution value for every prediction. Rooted in cooperative game theory, SHAP explains how each feature affects a prediction compared to a baseline, making it a powerful tool for both global and local explainability (Lundberg & Lee, 2017). For instance, in complex models like XGBoost or DNNs, where feature interactions are intricate and non-linear, SHAP provides clear insights into the direction and magnitude of each feature's impact. Visualizations of SHAP values allow to understand the overall importance of features as well as their effects on individual predictions, making the model's behavior more transparent and trustworthy.

Local Interpretable Model-Agnostic Explanations (LIME) takes a localized approach to explainability, focusing on individual predictions. Unlike model-

embedded techniques, LIME is model-agnostic, meaning it can be applied to any machine learning model. It works by perturbing the input data, observing how predictions change, and fitting a simple interpretable model, such as a linear regression, to approximate the behavior of the complex model locally (Ribeiro et al., 2016). For example, Zafar and Khan (2021) demonstrated the effectiveness of LIME in generating stable and deterministic explanations by modifying its random sampling approach to create consistent results. In their study, LIME was applied to understand the predictions of a classification model, where the local interpretations provided clear and actionable insights into how specific input features influenced individual decisions. This stability enhancement made LIME even more reliable and practical for applications requiring detailed decision-level insights. LIME complements global explainability techniques like SHAP by diving into the nuances of individual cases, making it a valuable tool for detailed decision-making.

Overall, Model-embedded feature importance provided a straightforward understanding of how features influenced overall predictions, while SHAP offered deeper insights into feature interactions and their contributions to predictions at both global and individual levels. LIME complemented these approaches by focusing on specific predictions, highlighting the model's reasoning in localized contexts. Together, these techniques enhanced the interpretability of the machine learning models used in this study. This multi-faceted approach not only builds trust in the predictions but also ensures that the models are actionable and transparent, making them more practical for real-world decision-making scenarios, especially in complex fields like project management.

CHAPTER 4

RESULTS

This chapter presents the findings of the study, structured into three main sections. The first section focuses on exploratory data analysis (EDA), where the dataset is examined through descriptive statistics, correlation analysis, and descriptive graphs such as histograms. The distribution of sprint success levels is also analyzed. The second section reports the Classification Results, including the steps taken to address data imbalance through resampling techniques, the optimization of model performance via hyperparameter tuning, and the final classification outcomes across multiple models. The third section explores Explainability, employing various techniques to interpret the classification models. Feature importance analysis embedded within the models, Shapley Additive Explanations (SHAP) and Local Interpretable Model-Agnostic Explanations (LIME) are applied to provide a comprehensive understanding of the factors influencing sprint success.

4.1 Exploratory data analysis

In this study, first, descriptive statistics, including the mean, median, standard deviation, and range for each feature, are analyzed. Correlation analyses are conducted, so potential relationships are identified between features and their influence on the target variable, Success Label. Also, Success Label is explained in detail with a formula and a bar graph is used to visualize the distribution. Histograms are created for each feature to illustrate distributions, variability, and any potential skewness. This comprehensive EDA forms a critical foundation for the subsequent

modeling and evaluation stages, ensuring the dataset is well-understood and appropriately prepared for analysis.

4.1.1 Descriptive analysis

To gain an initial understanding of the company dataset, a descriptive analysis is used. This analysis provides a statistical summary of numerical features, including metrics mean, standard deviation, minimum, maximum and quartile values. As a result of this analysis, Table 3 is created. The table provides an overview of the features used in the dataset to better understand what influences sprint success in agile projects. It summarizes various factors with statistics like averages, ranges, and how the values are distributed, giving us a clearer picture of the data.

One of the most important features is PBI Percentage, which indicates the proportion of planned Product Backlog Items (PBIs) completed during a sprint. On average, teams complete around 73% of their planned work, although the values range widely, from as low as 23% to as high as 100%. PBIs are created and discussed by team before sprint planning. Mostly they are the items for implementation of product's features. When the PBI Percentage is higher than the average, we can understand that team is focused on delivering planned features or value-driven tasks rather than addressing bugs or technical debt.

Another key feature is Bug Percentage, which shows the number of bugs relative to the total items completed in a sprint. On average, bugs make up about 27% of the workload. Some sprints have higher bug percentages, which might mean the team was focusing on fixing issues rather than delivering new features, or fixing technical debt, doing refactoring.

The Number of Items feature highlights how much work is typically undertaken during a sprint. Teams handle an average of 20 items per sprint, but the workload can range from just 4 items to as many as 47. Also, effort estimation of items can affect the number of items that undertaken to sprint. If high estimated items are planned, the number that team can plan may be lower than expected. Several features focus on the quality of documentation, which are essential for effective sprint planning. For example, the No Description Item Percentage reveals the proportion of items without adequate descriptions, with an average of 12%. In some sprints, this percentage reaches as high as 74%, which could lead to confusion or delays. On the other hand, the Sufficient Description Item Percentage, averaging around 37%, shows the percentage of items with clear and complete descriptions. These features show the importance of well-documented items for ensuring smooth workflows and avoiding misunderstandings of items by team members.

Bug reporting quality is also an important factor, measured by features like No Repro Steps Percentage and No System Info Percentage. These indicate the proportion of bugs that lack reproduction steps or system-related details, respectively. For instance, about 13% of bugs, on average, lack reproduction steps, and in some sprints, this number can be up to 100%. Similarly, around 41% of bugs are reported without system information, which can be the reason for setting environment to reproduce bug slower and more challenging. This suggests that improving bug documentation practices could significantly enhance sprint productivity.

The Acceptance Criteria Percentage feature highlights the percentage of PBIs with well-defined acceptance criteria. On average, 69% of PBIs have these criteria, which is a positive sign. However, there are still cases where PBIs without any

acceptance criteria. This can create uncertainty and make it harder for teams to meet expectations or deliver value effectively because the requirements are not defined or known.

Each bug item has a severity level. For instance, Sev1-2 Percentage, Sev3 Percentage, and Sev4 Percentage capture the proportion of high, medium, and low-severity bugs, respectively. High-severity bugs account for around 22% of all bugs, while medium-severity bugs are the most common, making up 54%. Low-severity bugs are less frequent, averaging about 9%. These shows teams mostly give importance to the bug items relatively have high severity, Low-severity bugs like Severity 4 have not cared too much.

Effort distribution is another critical aspect of sprint planning. Metrics such as Effort XS, S, M, L, XL, and XXL show how tasks are divided based on their estimated effort. In sprints the teams mostly get medium and large tasks. This suggests that most teams aim for a balance, focusing on tasks that are challenging but manageable. However, there are occasional sprints with a high proportion of XXL tasks time to time.

Finally, Team Size shows the number of team members involved in a sprint. Mostly teams consist of 7 members on average, with sizes ranging from 5 to 10. This consistency suggests that team size is unlikely to be a major factor influencing sprint success, even though its impact on workload in the sprints and communication efficiency within team is worth considering. It most likely because the teams decide on the amount of work to include a sprint based on their team size and plan accordingly. Therefore, estimation and the actual velocity may not be dependent to team’s size.

These features provide a full view of the company data, highlighting strengths and weaknesses in sprints The study aims to investigate these patterns using machine learning techniques to understand the drivers of sprint success and how teams can improve their processes.

Table 3. Descriptive Analysis of Company data

Feature	Mean	Std	Min	0,25	0,5	0,75	Max
PBI Percentage	72,86	18,86	23,08	59,46	73,68	88,24	100,00
Bug Percentage	27,14	18,86	0,00	11,76	26,32	405,41	76,92
Number of Items	19,67	9,28	4,00	13,00	18,00	25,00	47,00
No Description Item Per	11,96	15,60	0,00	0,00	6,67	16,67	73,68
Sufficient Description Item Per	36,68	20,66	0,00	21,43	35,71	50,00	100,00
No Repro Steps Per (Bug)	12,98	25,18	0,00	0,00	0,00	15,79	100,00
No System Info Per (Bug)	40,65	34,65	0,00	0,00	35,71	66,67	100,00
Acceptance Criteria Per (PBI)	68,65	31,47	0,00	43,48	80,00	100,00	100,00
Sev1-2 Per (Bug)	21,50	25,44	0,00	0,00	14,29	33,33	100,00
Sev 3 Per (Bug)	53,73	35,67	0,00	25,00	60,00	81,82	100,00
Sev 4 Per (Bug)	9,21	21,68	0,00	0,00	0,00	6,67	100,00
Effort XS Per	3,16	5,96	0,00	0,00	0,00	5,00	100,00
Effort S	19,99	15,42	0,00	6,90	16,67	32,00	60,87
Effort M	28,22	15,64	0,00	16,67	27,78	40,00	70,00
Effort L	25,36	15,33	0,00	16,00	26,09	36,84	68,75
Effort XL	14,20	14,43	0,00	0,00	10,53	23,53	57,14
Effort XXL	7,81	14,35	0,00	0,00	0,00	8,70	75,00
Team Size	7,39	1,22	5,00	7,00	7,00	8,00	10,00

Also, for public data, same descriptive analysis is made. As indicated by Table 4, some of the key statistics of public data such as the trends and variation of the features can be easily presented. The average value for the Sprint duration feature which means that it signifies the length of one sprint in days is 18.06, but it has a standard deviation which is very high of 163.51 (Table 4). This shows that there exists a huge range within which iterations can be completed with the lowest being 1

day and the highest reaching a number of 5873 days which is a quite an outlier. Such extreme values seem to suggest that the sprints duration could be exceedingly large or alternatively, the input of data could be incorrect and requires verification. The Number of issues at start time is interpreted as that number of issues allocated at the beginning of a sprint. The average is 8.59 and most tend to range between 2 and 11 as illustrated by the interquartile range. Nevertheless, the maximum value of 112 does show that there were some cases where a large number of issues were targeted in the very first step.

Table 4. Descriptive Analysis of Public data

Feature	Mean	Std	Min	0,25	0,5	0,75	Max
Sprint duration	18,06	163,51	1.00	8.00	14.00	17.00	5873.00
No. of issues at start time	8,58	11,82	1.00	2.00	5.00	11.00	112.00
No. of issues added	1,19	3,90	0.00	0	0	1.00	74.00
No. of issues removed	1,06	3,74	0.00	0	0	1.00	70.00
No. of to-do issues	7,06	10,26	0.00	1.00	4.00	9.00	99.00
No. of in-progress issues	0,99	1,66	0.00	0	0	1.00	12.00
No. of done issues	0,66	1,49	0.00	0	0	1.00	14.00
Scrum master	1,32	1,08	1.00	1.00	1.00	1.00	9.00
Scrum team members	3,73	3,53	1.00	1.00	3.00	5.00	29.00

The Number of issues added, and Number of issues removed as a result of the sprint plan contains low means of 1.20 and 1.07 respectively. The range for these figures is also not very high. Their medians and the 75th percentiles are both 0 and 1, and this implies that most sprints recorded hardly experienced any change in scope. But the bigger figures ‘74’ and ‘70’ speak of cases where such changes to the contents of the sprint backlog were considerable and this could mean that there could be big changes at sprint goals. The Number of to-do issues has a mean of 7.07 and a standard deviation of 10.26, with most sprints having between 1 and 9 to-do items by

the prediction time. However, a maximum of 99 to-do issues points to potential inefficiencies or poorly scoped sprints in a few cases. Similarly, the Number of in-progress issues and Number of done issues have low means (0.99 and 0.66, respectively), reflecting limited activity in these categories for most sprints. However, their maximum values of 12 and 14 indicate a small subset of sprints with higher activity.

The Scrum master feature, with a mean of 1.32 and limited variability (IQR: 1–1), suggests that most sprints consistently involve one Scrum master, as expected. However, a maximum value of 9 could indicate rare cases where additional administrative support or overlapping Scrum master roles were required. Considering members of a Scrum team as a feature, average at mean of 3.73 and standard deviation of 3.54, indicates moderate variability within the size of team members. Most teams are small teams having IQR of 5 members, while some Sprints can have a maximum of 29 members suggesting those sprints are more complicated or more than one team participates in the same sprint and work collaboratively which in turn complicates the processes involved in those sprints.

4.1.2 Correlation analysis

The Correlation Analysis acts as a pictorial scheme showing the extent to which certain notable features present in the proprietary dataset interact (form relationships) within a basket of correlation values of between negative one (-1) and one (1). A correlation value of one (1), for example, is indicative of a direct relationship whereby the features increase or decrease together, while negative one (-1) indicates that as one feature increases the other reduces. Values which are around zero suggest

that there is almost no linear relationship between the features. Correlation analysis was conducted for the company data (Table 5).

The PBI Percentage and Bug Percentage exhibit a perfectly negative correlation (-1.000), reflecting their complementary nature in the dataset. As the proportion of PBIs increases, the proportion of bugs decreases proportionally. Both features show moderate to strong correlations with other metrics, such as Number of Items (positive for Bug Percentage at 0.489 and negative for PBI Percentage at -0.489) (Table 5). This indicates that iterations with a higher number of items tend to involve a greater proportion of bugs.

The correlation analysis gives a more detailed picture of the interplay of the different sprint measures, and it also shows a few other intriguing observations. There is a moderate positive correlation (0.489) between ‘Bug Percentage’ and ‘Number of Items’ which tells us that more tasks in a sprint give rise to more bugs (Table 5). This points to the fact that with an increase in the number of deliverables, the quality may suffer, and, in this case, it can be due to insufficient manpower or less attention to detail in testing. The Number of Items correlates positively with Team Size (0.500), suggesting that larger teams handle more items in an iteration (Table 5).

There are some weak correlations present in the dataset. For instance, the correlation between ‘Acceptance Criteria Per (PBI)’ and ‘Number of Items’ (0.057) indicates no significant association between how well-articulated acceptance criteria influence the total effort in a sprint. Similarly, the correlation between ‘Effort XS Per’ and ‘No System Info Per (Bug)’ (-0.143) suggests that the lack of systemic information in bug reports is not strongly affected by small effort estimations.

In contrast, stronger correlations can be observed in specific cases. For example, the correlation between ‘Effort L Per’ and ‘Sev1-2 Per (Bug)’ (0.376) indicates that higher effort estimates are likely to be associated with a greater number of high-severity bugs (Table 5). These findings highlight how different effort levels may influence the severity of bugs in a sprint.

This analysis demonstrates how complex the interrelationships in the dataset are and is the first step towards enumerating the factors that will be useful in forecasting success in a sprint. Thus, creating a foundation for further statistical and machine learning quantitative analyses.

For public data, the correlation analysis highlights the relationships between sprint duration, team composition, and issue management (Table 6). When considering No. of issues at start time (-0.902), Number of issues added (-0.336), Number of issues removed (0.462), it is possible to note that most features have weak to moderate negative correlation with sprint duration (Table 6). This means that in cases where issue sprints are long, there is not a very strong link to a considerable number of issues being resolved, which means that the duration of a sprint may not be an important determinant of the number of issues that would be added or removed. The Number of issues at start time exhibits strong positive correlations with features such as Number of issues added (0.193), Number of issues removed (0.441), and Number of to-do issues (0.929) (Table 6). This relationship suggests that iterations starting with a larger number of issues tend to involve higher activity in terms of adding or resolving issues, as well as having more to-do items by the prediction time. Similarly, Number of to-do issues correlates positively with Number of in-progress issues (0.342) and Number of done issues (0.328), indicating that larger to-do lists are associated with higher activity during the sprint (Table 6).

Table 5. Correlation Analysis of Company Data

	PBI Percentage	Bug Percentage	Number of items	No Description Item Per	Sufficient Description Item Per	No Repro steps Per (Bug)	No System info Per (Bug)	Acceptance Criteria Per (PBI)	Team Size	Sev 1-2 Per (Bug)	Sev 3 Per (Bug)	Sev 4 Per (Bug)	Effort XS	Effort S	Effort M	Effort L	Effort XL	Effort XXL
PBI Percentage	1,000	-1,000	-0,489	0,195	0,223	0,065	-0,122	-0,081	-0,238	-0,300	-0,438	0,041	0,096	0,333	0,126	-0,307	-0,313	0,151
Bug Percentage	1,000	1,000	0,489	-0,195	-0,223	-0,065	0,122	0,081	0,238	0,300	0,438	-0,041	-0,096	-0,333	-0,126	0,307	0,313	-0,151
Number of items	-0,489	0,489	1,000	-0,027	-0,225	0,013	0,226	0,057	0,500	0,130	0,302	-0,048	-0,177	-0,225	-0,084	0,119	0,399	-0,168
No Description Item Per	0,195	-0,195	-0,027	1,000	-0,276	0,106	0,010	-0,477	-0,437	0,051	-0,220	0,116	-0,106	-0,023	0,187	-0,089	0,133	-0,157
Sufficient Description Item Per	0,235	-0,223	-0,225	-0,276	1,000	0,134	0,107	0,051	0,203	-0,238	-0,012	-0,068	-0,060	-0,013	-0,206	-0,214	-0,006	0,496
No Repro steps Per (Bug)	0,065	-0,065	0,013	0,106	0,134	1,000	0,465	-0,057	0,035	-0,099	-0,002	0,488	0,212	0,078	0,136	-0,283	-0,038	0,012
No System info Per (Bug)	-0,122	0,122	0,226	0,010	0,107	0,465	1,000	-0,180	0,085	-0,017	0,343	0,292	0,063	-0,068	0,048	0,050	0,061	-0,143
Acceptance Criteria Per (PBI)	-0,081	0,081	0,057	-0,477	0,051	-0,057	-0,180	1,000	0,334	-0,228	0,044	0,101	0,212	0,104	-0,011	-0,188	-0,217	0,217
Team Size	-0,238	0,238	0,500	-0,437	0,203	0,035	0,085	0,334	1,000	-0,078	0,099	-0,003	-0,147	-0,253	-0,282	-0,085	0,277	0,404
Sev 1-2 Per (Bug)	-0,300	0,300	0,131	0,052	-0,238	-0,099	-0,017	-0,229	-0,079	1,000	-0,234	-0,187	-0,011	-0,188	-0,121	0,376	0,168	-0,240
Sev 3 Per (Bug)	-0,438	0,438	0,302	-0,221	-0,012	-0,003	0,344	0,044	0,100	-0,234	1,000	-0,287	-0,191	-0,221	-0,071	0,220	0,166	-0,045
Sev 4 Per (Bug)	0,042	-0,042	-0,048	0,116	-0,069	0,488	0,293	0,101	-0,004	-0,187	-0,287	1,000	0,265	0,231	0,254	-0,204	-0,261	-0,145
Effort XS	0,096	-0,096	-0,177	-0,107	-0,061	0,213	0,063	0,213	-0,148	-0,011	-0,191	0,265	1,000	0,252	0,010	-0,184	-0,322	-0,152
Effort S	0,334	-0,334	-0,225	-0,023	-0,013	0,078	-0,068	0,104	-0,253	-0,188	-0,221	0,231	0,252	1,000	0,297	-0,489	-0,634	-0,342
Effort M	0,127	-0,127	-0,084	0,187	-0,206	0,136	0,049	-0,012	-0,282	-0,121	-0,071	0,254	0,010	0,298	1,000	-0,349	-0,522	-0,504
Effort L	-0,307	0,307	0,120	-0,089	-0,215	-0,283	0,051	-0,189	-0,085	0,376	0,220	-0,204	-0,184	-0,489	-0,349	1,000	0,141	-0,220
Effort XL	-0,314	0,314	0,399	0,134	-0,006	-0,038	0,061	-0,217	0,277	0,167	0,165	-0,260	-0,321	-0,634	-0,521	0,141	1,000	0,228
Effort XXL	0,152	-0,152	-0,168	-0,157	0,496	0,012	-0,143	0,217	0,404	-0,240	-0,045	-0,145	-0,151	-0,342	-0,503	-0,200	0,228	1,000

Scrum team members in fact display moderate positive linear relations with Number of issues at start time (0.564), Number of issues added (0.347), and Number. of issues removed (0.339), which is quite interesting (Table 6). This indicates the fact that in a larger team, the volume of issues that is intervened during sprints is likely to be more. On the other hand, Scrum master has weak relations with most of the features which means that the number or existence of scrum masters do not have significant role in the management of issues or sprint results.

Table 6. Correlation Analysis of Public Data

	Iteration duration	No. of issues at start time	No. of issues added	No. of issues removed	No. of to-do issues	No. of in-progress issues	No. of done issues	Scrum master	Scrum team members
Iteration duration	1,000	-0,009	-0.003	0,005	-0,009	-0,015	-0,011	-0,010	-0,016
No. of issues at start time	-0,009	1,000	0,192	0,441	0,929	0,453	0,433	0,002	0,564
No. of issues added	-0,003	0,192	1,000	0,468	0,362	0,229	0,218	-0,049	0,347
No. of issues removed	0,005	0,441	0,468	1,000	0,289	0,126	0,100	-0,047	0,339
No. of to-do issues	-0,009	0,929	0,362	0,289	1,000	0,342	0,328	0,010	0,537
No. of in-progress issues	-0,015	0,453	0,229	0,126	0,342	1,000	0,440	-0,076	0,453
No. of done issues	-0,011	0,433	0,218	0,100	0,328	0,440	1,000	0,019	0,322
Scrum master	-0,010	0,002	-0,049	-0,047	0,010	-0,076	0,019	1,000	0,014
Scrum team members	-0,016	0,564	0,347	0,339	0,537	0,453	0,322	0,014	1,000

Overall, the analysis highlights that the activities with regard to issues are strongly related to the number of issues at the beginning of the sprint and the size of the Scrum team. On the contrary, time of a sprint and the functionality of Scrum

masters are observed to have low correlation, indicating that perhaps other parameters may be more significant in explaining sprint efficiency and results.

4.1.3 Success levels of sprints

Iterations are distributed across three success levels based on the percentage of planned workload completed for both company and public data. Success is categorized into three levels: Low, Medium, and High. Company data can be shown at Figure 1.

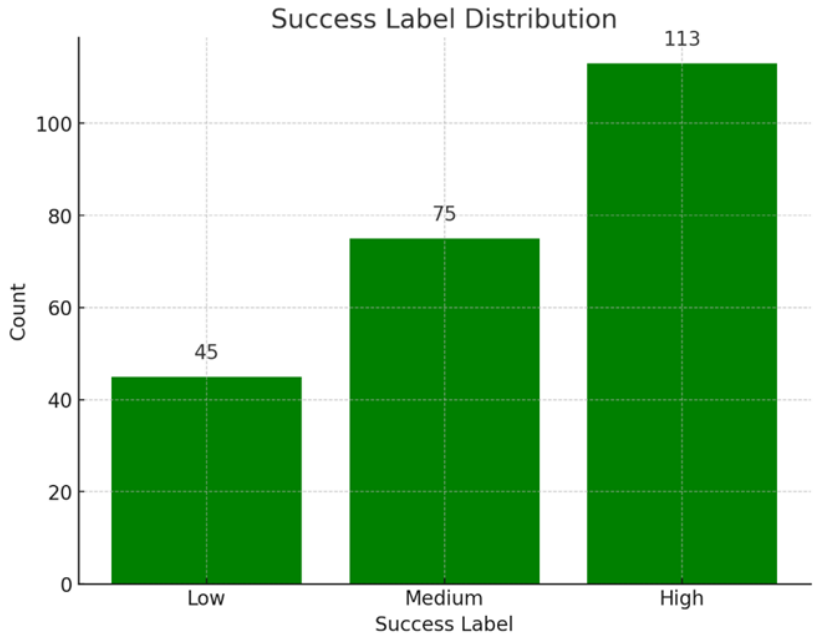

Figure 1. Success label distribution of company data

The High Success category, representing iterations where 75% or more of the planned workload was completed, is the most common with 113 iterations .This indicates that most sprints are very productive, with teams consistently meeting or exceeding their goals.The Medium Success category, where 50–75% of the planned

workload was achieved, includes 75 iterations. This suggests that a good portion of sprints perform reasonably well, even if they don't hit the highest success threshold. Finally, the Low Success category—iterations completing less than 50% of the planned workload—is the smallest group, with just 45 iterations. While relatively rare, these sprints highlight instances where teams may have faced significant challenges or misaligned expectations.

Overall, the distribution highlights a positive trend in the company's sprint performance, with most iterations achieving at least 50% of their planned workload. However, understanding the reasons of Low Success iterations could help the company to increase sprint performances.

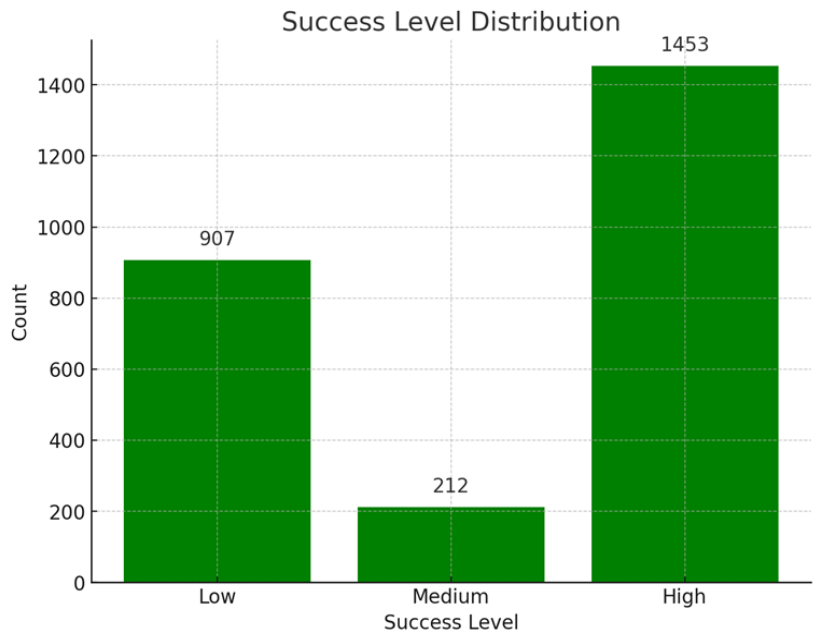

Figure 2. Success label distribution of public data

Figure 2 shows the success level distribution of iterations at public data. The High Success category, where 75% or more of the planned workload was completed,

dominates the distribution with 1,453 iterations. The Low Success category, representing iterations with less than 50% of the planned workload achieved, accounts for 907 iterations. Finally, the Medium Success category, where 50–75% of the planned workload was completed, is the least frequent with 212 iterations. This distribution points out that most of the iterations are found within the limit of High and Low Success, with relatively few in the medium level.

4.1.4 Histograms

Histograms are an efficient approach to visualizing the distributions of the values of features, and so they serve as an efficient exploratory data analysis tool. They provide an overview of key features in the dataset, highlighting important patterns and trends. Histograms were created for all features in the company data. The PBI Percentage histogram further reveals that a large number of iterations has a lot of PBIs, reaching an extreme at 100 percent. It means that some sprints only focus on PBIs with little or no bug fixing tasks. In contrast, the Bug Percentage histogram shows a concentration at lower values, suggesting that bugs form a smaller part of the workload in most iterations, reflecting a general emphasis on new development tasks. The Number of Items histogram exhibits a roughly normal distribution, peaking around 15–25 items per iteration. This suggests a consistent workload allocation across sprints, with most iterations handling a moderate number of tasks (Figure 3).

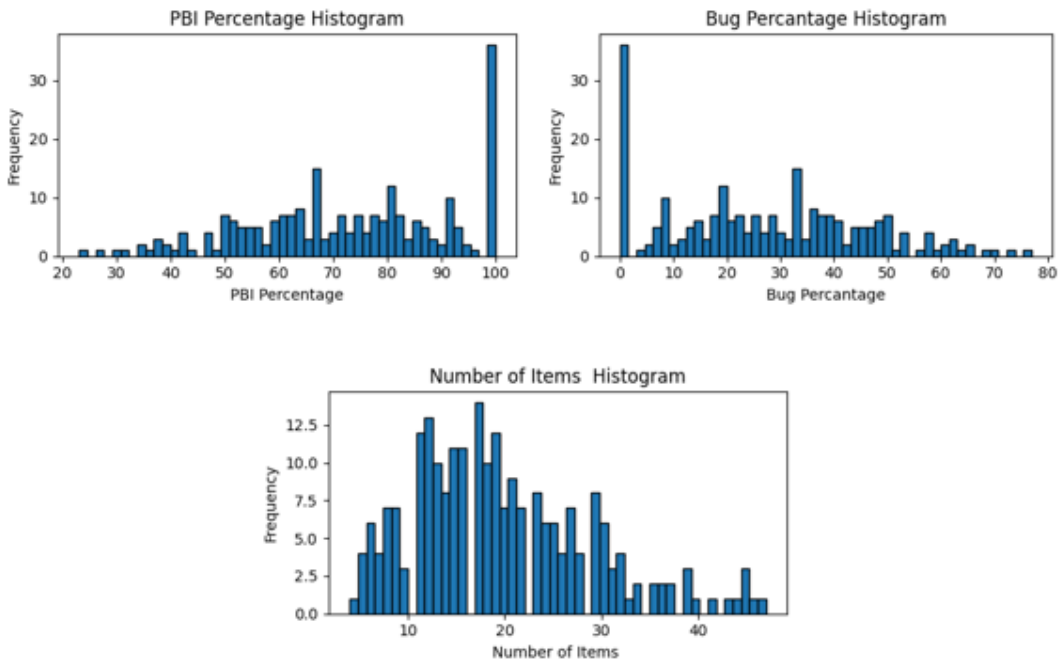

Figure 3. Histograms of item types and distributions

On the other hand, the No Description Item Percentage is heavily skewed toward 0, indicating that the majority of tasks are well-documented. The lack of descriptions for some items indicates that these items are not actually ready to be included in the sprint. The Sufficient Description Item Percentage contains a wide range of values without any strong clustering, indicating inconsistency on how tasks are documented across iterations. The No Repro Steps Per (Bug) and No System Info Per (Bug) histograms highlight that many bugs lack critical information, as these distributions are heavily skewed toward higher values, suggesting potential inefficiencies in bug reporting practices. Acceptance Criteria Per (PBI) histogram shows a strong peak at 100%, indicating that a significant portion of PBIs have clear acceptance criteria, which is crucial for ensuring task clarity and completion standards (Figure 4).

The Sev1-2 Per (Bug) histogram is heavily skewed toward lower values, indicating that high-severity bugs (Sev1-2) are only a small proportion in most iterations. Even though, some iterations report higher percentages occasionally, this means teams get critical bugs to the iterations time to time. The Sev 3 Per (Bug) histogram, on the other hand, has a broader spread, with many iterations containing 50–100% Sev3 bugs, indicating that moderate-severity bugs frequently dominate iterations. Similarly, the Sev 4 Per (Bug) histogram is concentrated near 0%, showing that low-severity bugs (Sev4) are relatively rare and typically a minor part of iterations (Figure 5).

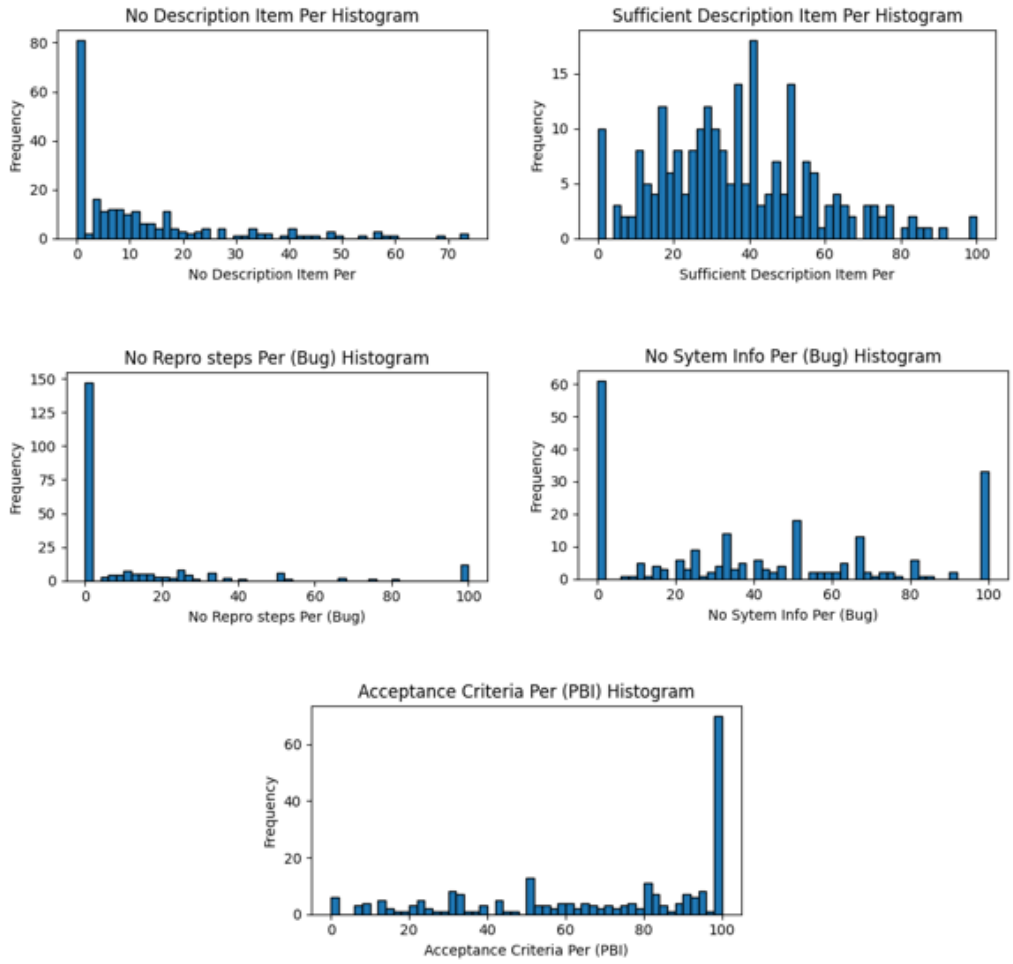

Figure 4. Histograms of key characteristics of well-documented items

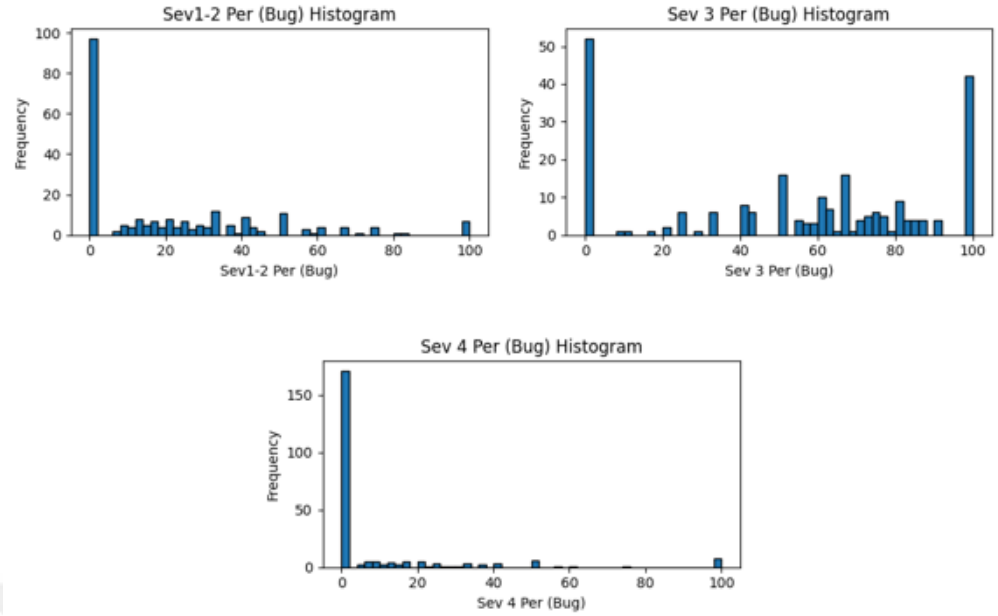

Figure 5. Histograms of severity distributions

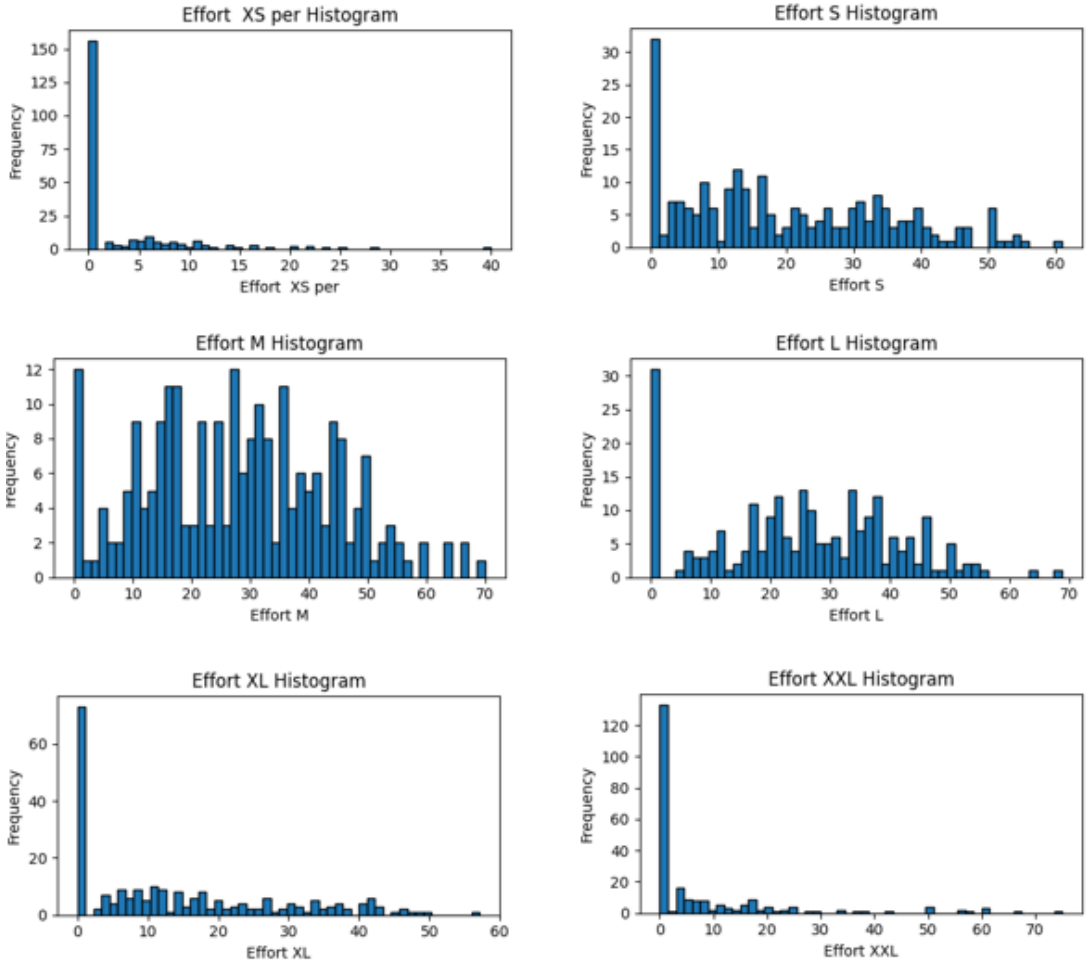

Figure 6. Histograms of effort distributions

The effort histograms presenting some interesting patterns. The Effort XS Per histogram is skewed toward lower values, indicating that extra small tasks are infrequent. However, Effort S, Effort M, and Effort L show more even distributions, reflecting that these effort levels are common and spread across sprints. The Effort XL Per histogram is skewed toward smaller values, showing that tasks with higher efforts are less common, while the Effort XXL Per histogram is heavily skewed toward 0%, confirming that tasks with highest efforts are rare in most sprints and only occur occasionally in higher percentages (Figure 6).

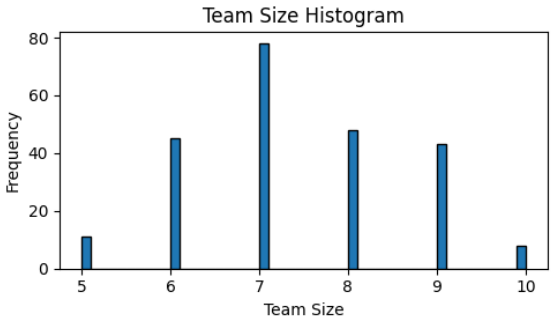

Figure 7. Histogram of team size

The “Team Size” histogram shows that most teams consist of 6 to 9 members, with a peak at 7. Teams with 5 or 10 members are less common, indicating a preference for medium-sized teams. This aligns with common agile practices, where team sizes are optimized for collaboration and productivity (Figure 7).

Overall, these histograms highlight some significant trends across the key features of the dataset especially in the patterns of task distribution, bug severity, amount of effort, or team structure. The findings suggest a general focus on well-documented tasks, balanced team sizes, and moderate effort levels, with occasional outliers requiring further attention.

For the features of public data, histograms are created to provide valuable insights into key metrics related to sprint performance and project management in Scrum. The histogram for "Planday" (Sprint Duration) shows that the majority of iterations are short, with a significant drop-off as the duration increases (Figure 8). There is a larger trend in the environment that indicates that shorter sprints are preferred, which resonates with the Agile emphasis on getting feedback and making changes on a frequent basis.

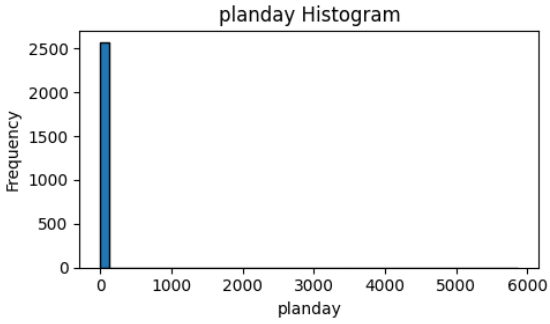

Figure 8. Histogram of sprint duration

The "No Issue Start Time" histogram highlights that most issues are initiated early in the sprint, with a gradual decline thereafter (Figure 9). This suggests that teams prioritize tasks effectively and address them promptly at the beginning of the sprint. Similarly, we also see distributions for “No Issue Added” and “No Issue Removed” being close to each other, confirming that modifications made to project scope during a sprint are not excessive, thus reinforcing the rational way Sprint planning is done, where the teams have a certain scope to focus on, in order to prevent the expansion of scope.

The "No Issue To-Do" histogram shows that most teams have a small number of to-do items, though some sprints have higher task loads, reflecting variations in team capacity or task complexity. Additionally, the "No Issue In Progress" data

indicates that teams typically have very few tasks in progress at any given time, adhering to Work-In-Progress (WIP) limits to maintain focus and reduce multitasking. The ‘No Issue Done’ histogram indicates that during a single sprint most teams succeeded in completing only a limited number of tasks which could be attributed to the limited duration for a sprint or the breaking of the tasks into smaller components (Figure 9).

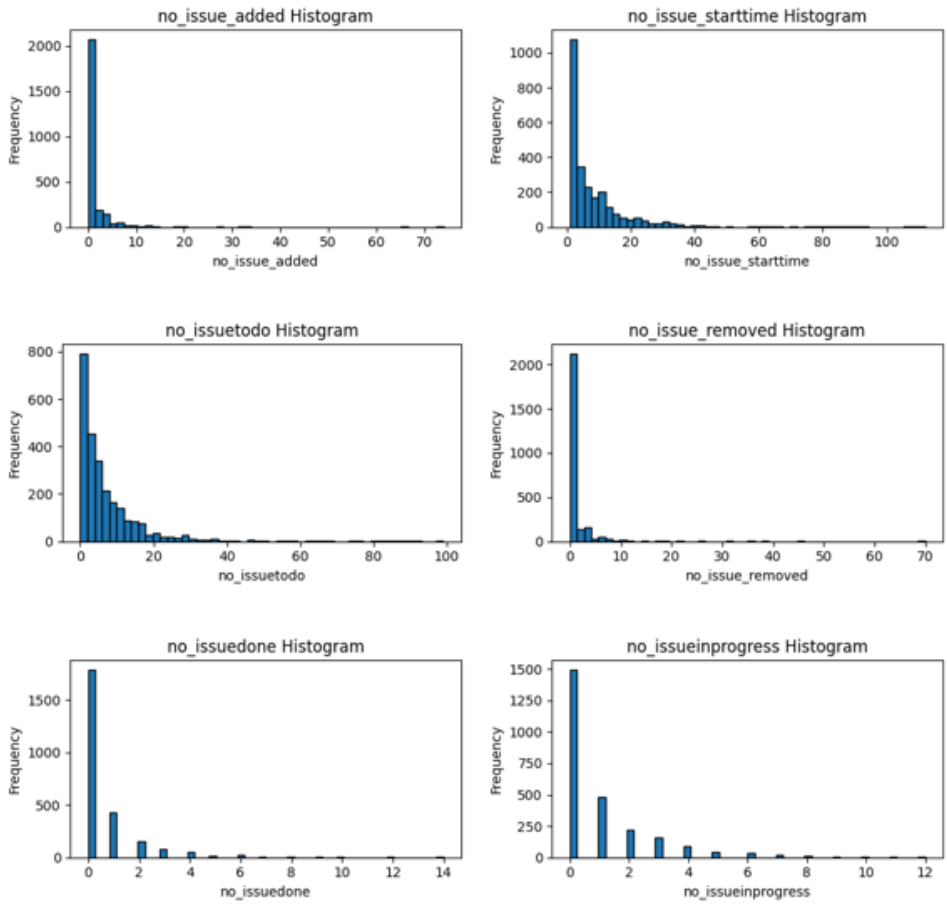

Figure 9. Histograms of items' status and transition distributions

Regarding team structure, the "No Admin" (number of Scrum Master in a team) histogram shows that most teams operate with a single Scrum Master, consistent with standard Agile practices where one Scrum Master facilitates multiple

teams. It should be noted that a similar situation can be observed with the "No Team Member" histogram, here too, the main concentration is in the smaller members, mainly in the range 5-9, corresponding with the range suggested by Scrum on the ideal size of cross-functional teams for better collaboration (Figure 10).

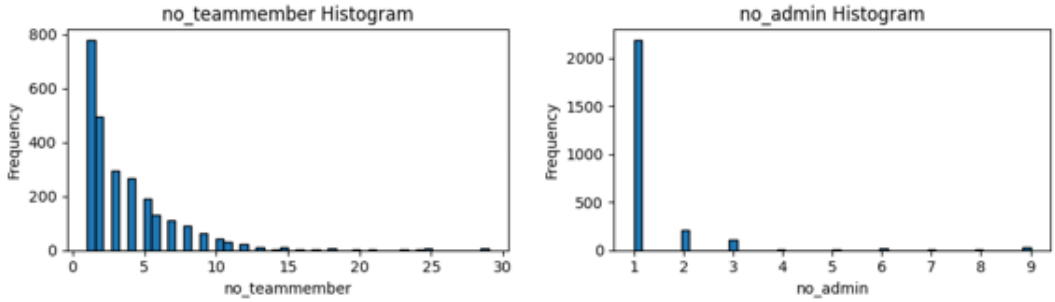

Figure 10. Histogram of team structure

Overall, the data reflects key characteristics of well-implemented Scrum practices. These include a disciplined approach to sprint planning, minimal scope changes, adherence to WIP limits, and the use of small, cross-functional teams. These trends support the basic principles of focus, commitment, and incrementalism as key to the practice of Agile and Scrum approaches.

4.2 Resampling

In both the company and public datasets, the target variable "Success Label" exhibited significant class imbalance, where certain classes were overrepresented while others had very few instances. For instance, in the company dataset, the class distribution for "High," "Medium," and "Low" success labels was 113, 75, and 45 instances, respectively. Similarly, the public dataset showed an even more pronounced imbalance, with 291 instances for "High," 182 for "Low," and only 42

for "Medium." This imbalance posed a challenge for classification models, as they tend to favor the majority class, resulting in biased predictions and poor performance on underrepresented classes. To address this issue, the Synthetic Minority Oversampling Technique (SMOTE) is used to balance the class distributions in the training data. SMOTE creates synthetic samples for minority classes by interpolating between existing instances, rather than merely duplicating data points. This approach guarantees that the resampled data is not only balanced but also diverse, preserving the overall structure of the dataset. This method was applied only to the training data, leaving the test set unchanged to maintain its original distribution for realistic evaluation.

The resampling process began with stratified train-test splitting to maintain the original class distribution in the test set. After SMOTE is applied, the training data became a perfectly balanced class distributed data, with all classes having the same number of instances. For example, in the company dataset, the "Low," "Medium," and "High" classes were all resampled to match the size of the majority class (113 instances). Similarly, in the public dataset, SMOTE adjusted the training data so that all classes were balanced to the size of the largest class. Balancing the training data allowed the model to learn from all classes effectively, leading to more accurate predictions. This adjustment was especially important to avoid bias toward the majority classes and ensure that the model performed well across all success levels.

4.3 Hyperparameter tuning

The hyperparameter tuning process was conducted systematically using Grid Search and Randomized Search methods to find the best configurations for both datasets.

The same parameter ranges and methods were applied across both datasets to ensure consistency and comparability. The final results give the optimal parameters for each model, which achieved the best performance on their respective datasets. The best-performing parameters are marked with asterix in the Table 7.

For Logistic Regression, the parameter tuning involved C ([0.1, 1, 10, 100]), penalty (['l2']), and solver (['sag', 'lbfgs', 'saga', 'liblinear']). The optimal configuration, with C=1 and the lbfgs solver, demonstrated the importance of proper regularization and an efficient optimization algorithm for balancing bias and variance (Table 7).

The Decision Tree model performed best with a max_depth of 20, max_features set to sqrt, and min_samples_split of 5 (Table 7). These parameters suggest a moderately deep tree that captures sufficient data complexity without overfitting. Similarly, for the Random Forest, the best configuration included 200 estimators, a maximum depth of 20, and bootstrap=True. These settings ensured a robust ensemble with diverse trees, enhancing model stability and accuracy.

For SVM, the best results were achieved with C=10, kernel='rbf', and gamma='scale' (Table 7). This configuration reflects the strength of the radial basis function (RBF) kernel in handling non-linear relationships within the data while maintaining regularization to prevent overfitting.

In the case of XGBoost, the optimal parameters included a learning rate (eta) of 0.1, gamma=0.5, and a maximum depth of 4. These settings balanced the model's learning capability and complexity, effectively preventing overfitting while maintaining sufficient flexibility for learning intricate patterns.

Table 7. Hyperparameter Tuning

Model	Hyperparameter	Value	Method
Logistic Regression	C	[0.1, 1*, 10, 100]	Grid Search
	penalty	‘l2*'	
	solver	['sag', 'lbfgs*', 'saga', 'liblinear']	
Decision Tree	max_depth	[10, 20*, 30]	Grid Search
	max_features	['sqrt'*, 'log2']	
	min_samples_split	[2*, 5, 10]	
	min_samples_leaf	[1*, 2, 4]	
	criterion	['gini'*, 'entropy']	
RF	n_estimators	[100, 200, 300*]	Grid Search
	max_features	['sqrt'*, 'log2']	
	max_depth	[10, 20*, 30]	
	min_samples_split	[2*, 5, 10]	
	min_samples_leaf	[1*, 2, 4]	
	bootstrap	[True, False*]	
SVM	C	[0.1, 1*, 10, 100]	Grid Search
	kernel	['linear', 'rbf'*, 'poly', 'sigmoid']	
	gamma	['scale'*, 'auto']	
	degree	[2, 3*, 4]	
	coef0	[0.0*, 0.1, 0.5]	
XGBoost	eta	[0.3*, 0.1, 0.5]	Grid Search
	gamma	[0*,0.5, 1, 2]	
	max_depth	[3, 4, 5,6*]	
	min_child_weight	[1*,3, 5]	
Gradient Boosting	learning_rate	[0.025, 0.05, 0.1*, 0.2, 0.3]	Grid Search
	max_depth	[2, 3*, 5, 7, 10, None]	
	min_samples_split	[2*, 5, 10, 20]	
	max_features	['log2', 'sqrt', 0.25, 1.0,None*]	
	subsample	[0.15, 0.5, 0.75, 1.0*]	
LightGBM	learning_rate	[0.025, 0.05, 0.1, 0.2, 0.3*]	Grid Search
	num_leaves	[3, 7, 15*, 31, 127, 1024]	
	top_rate	[0.2*, 0.4, 0.6, 0.7]	
	other_rate	[0.05*, 0.1, 0.3]	
	feature_fraction_bynode	['log2', 'sqrt', 0.25, 1.0*]	
DNN	units_layer1	[64, 128, 256*]	Grid Search
	dropout_layer1	[0.1, 0.3, 0.5*]	
	units_layer2	[32, 64, 128*]	
	dropout_layer2	[0.1, 0.3, 0.5*]	
	learning_rate	[0.001*, 0.0005, 0.0001]	
FCNN	model__units_1	[32, 64, 128*]	Grid Search
	model__units_2	[16, 32, 64*]	
	model__dropout_1	[0.2, 0.3*, 0.4]	
	model__dropout_2	[0.2, 0.3*, 0.4]	
	model__learning_rate	[0.01*, 0.001, 0.0001]	
	batch_size	[16*, 32]	
	epochs	[50, 100*]	

Similarly, Gradient Boosting performed best with a learning rate of 0.05, a maximum depth of 5, and a subsample rate of 0.75 (Table 7). This configuration

emphasized gradual learning with a focus on stability and generalization. For LightGBM, the best-performing parameters were a learning rate of 0.05, num_leaves=31, and a top_rate of 0.4 (Table 7). These parameters highlight LightGBM's ability to efficiently handle large datasets with categorical and numerical features, delivering high accuracy while being computationally efficient.

The deep learning models, DNN and FCNN, were tuned for parameters such as the number of neurons in hidden layers (units_layer1 and units_layer2), dropout rates (dropout_layer1 and dropout_layer2), and the learning rate. For the FCNN model, the best configuration included 128 neurons in the first layer, 64 neurons in the second layer, dropout rates of 0.3 for both layers, and a learning rate of 0.001 (Table 7). These settings provided the model with strong generalization capabilities, especially for handling the classification of success levels in Scrum.

The hyperparameter tuning process provided meaningful insights into how each model performs and how their settings can be adjusted to better match the specific characteristics of the data. By finding the best configurations, the models were able to deliver strong and reliable results on both the company dataset and the public dataset. This process really shows how important it is to fine-tune models carefully to get the most accurate and reliable outcomes. The best results will be presented at classification results.

4.4 Classification results

The classification results provide a detailed view of how various machine learning and deep learning models performed in predicting sprint success levels (Low, Medium, High) within the datasets. These models were first applied to the company dataset (Table 8). Overall, the models performed better for the Low Success and

High Success classes, while the Medium Success class consistently exhibited the lowest F1-scores. This shows that Medium Success is the most difficult category to classify, perhaps because of its complex nature and lower representation in the data.

Table 8. Classification Results of Company Data

Model	Success Level	Precision	Recall	F1-score
Logistic Regression	1	0.26	0.55	0.35
	2	0.31	0.33	0.32
	3	0.75	0.39	0.51
	Weighted average	0.51	0.40	0.42
Decision Tree	1	0.50	0.55	0.52
	2	0.41	0.46	0.43
	3	0.55	0.47	0.51
	Weighted average	0.49	0.48	0.49
SVM	1	0.40	0.66	0.50
	2	0.50	0.46	0.48
	3	0.77	0.60	0.68
	Weighted average	0.61	0.57	0.58
Gradient Boosting	1	0.45	0.55	0.50
	2	0.37	0.40	0.38
	3	0.75	0.65	0.69
	Weighted average	0.57	0.55	0.56
LightGBM	1	0.50	0.77	0.60
	2	0.33	0.26	0.29
	3	0.57	0.52	0.54
	Weighted average	0.48	0.48	0.47
RF	1	0.70	0.77	0.73
	2	0.43	0.46	0.45
	3	0.66	0.60	0.63
	Weighted average	0.59	0.59	0.59
XGBoost	1	0.63	0.77	0.70
	2	0.30	0.26	0.28
	3	0.65	0.65	0.65
	Weighted average	0.53	0.55	0.54
DNN	1	0.30	0.33	0.32
	2	0.36	0.33	0.34
	3	0.70	0.70	0.70
	Weighted average	0.45	0.45	0.45
FCNN	1	0.54	0.78	0.64
	2	0.42	0.33	0.37
	3	0.73	0.70	0.71
	Weighted average	0.56	0.60	0.57

Among the traditional machine learning models, Logistic Regression and Decision Tree showed limited performance. Logistic Regression achieved an F1-score of 0.38 for Low Success and 0.51 for High Success but struggled significantly with Medium Success, obtaining an F1-score of just 0.32 (Table 8). Decision Tree performed slightly better, achieving its best F1-score of 0.52 for Low Success, but its performance for Medium and High Success was moderate, with F1-scores of 0.43 and 0.51, respectively (Table 8). These results suggest that traditional models struggled to capture the complexity of relationships in the data, especially for the Medium Success class.

SVM demonstrated more balanced performance, performing well for High Success with an F1-score of 0.68 and achieving moderate results for Low Success (0.50). However, it still struggled with Medium Success (0.48), indicating that this class presents a consistent challenge across all models. Similarly, SVR showed reasonable results, achieving F1-scores of 0.56 for both Low and High Success, while performing slightly better for Medium Success (F1-score of 0.50) compared to other traditional models (Table 8).

The ensemble methods, particularly Random Forest and XGBoost, delivered the best overall performance. Random Forest achieved the highest F1-score for Low Success at 0.73 and performed well for High Success with an F1-score of 0.63. XGBoost also demonstrated consistent performance, with F1-scores of 0.70 for Low Success and 0.65 for High Success. However, both models struggled with Medium Success, achieving F1-scores of 0.45 (Random Forest) and 0.28 (XGBoost), indicating room for improvement in handling this class. Gradient Boosting performed well for High Success with an F1-score of 0.69 but struggled overall, especially for Medium Success (0.38). LightGBM underperformed compared to

other ensemble methods, with its highest F1-score being 0.54 for High Success and lower scores for the other classes.

Among the deep learning models, FCNN emerged as the strongest performer, achieving F1-scores of 0.64 for Low Success and 0.71 for High Success. It outperformed DNN, which achieved a comparable F1-score of 0.70 for High Success but struggled significantly with Low Success (0.32) and Medium Success (0.34).

These results suggest that FCNN was better able to capture complex patterns in the data, particularly for the Low and High Success classes. In conclusion, Random Forest, XGBoost, and FCNN demonstrated the most balanced and reliable performance, particularly excelling in predicting Low and High Success levels. However, Medium Success remains a consistent challenge across all models, highlighting the need for further refinement in feature engineering, resampling strategies, or advanced modeling techniques to improve classification for this category. Strengths and weaknesses of different models are found with these analyses.

Same models are applied for public data to predict sprint success level (Table 9). Logistic Regression demonstrated weak overall performance, particularly for Medium Success, where it achieved an F1-score of only 0.04. For Low Success, the model performed moderately with an F1-score of 0.63, and for High Success, it achieved a slightly better F1-score of 0.64 (Table 9). These results reflect the limitations of Logistic Regression in handling the complexity of the data and effectively classifying nuanced success levels. Decision Tree, on the other hand, performed significantly better, achieving its best result for High Success with an F1-score of 0.80. It also performed well for Low Success (F1-score of 0.72), but Medium Success remained challenging with an F1-score of 0.49. The model's

relatively high recall for High Success (0.74) indicates its ability to correctly identify most instances of this class, but its performance for Medium Success not very well (Table 9).

SVM struggled significantly across all success levels, especially for Medium Success, where it achieved a very low F1-score of just 0.18 (Table 9). Its performance for Low Success was slightly better, with an F1-score of 0.57, and for High Success, it managed a score of 0.43 (Table 9). These results highlight SVM's difficulty in handling the complexities and imbalances present in the public dataset. Compared to other models, SVM struggled to deliver competitive results.

Ensemble methods, including Random Forest, LightGBM, and XGBoost, clearly show strong and consistent performance. LightGBM, as one of the best-performing models, achieved an F1-score of 0.83 for High Success and 0.78 for Low Success (Table 9). However, its performance for Medium Success was weaker, with an F1-score of 0.52. Random Forest also performed well, achieving F1-scores of 0.83 for High Success and 0.76 for Low Success, with a better result for Medium Success (F1-score of 0.61), making it one of the most balanced models (Table 9). XGBoost showed similar robustness, achieving an F1-score of 0.80 for High Success and 0.72 for Low Success, although its Medium Success F1-score of 0.59 lagged slightly behind Random Forest (Table 9).

Gradient Boosting achieved strong results, particularly for High Success, with an F1-score of 0.81. For Low Success, it achieved an F1-score of 0.73, while its performance for Medium Success was weaker, with an F1-score of 0.54 (Table 9). These results emphasize its general ability in analyzing the dataset, although the problems with Medium Success remained.

Table 9. Classification Results of Public Data

Model	Success Level	Precision	Recall	F1-score
Logistic Regression	1	0.54	0.77	0.63
	2	0.04	0.04	0.04
	3	0.77	0.56	0.64
	Weighted average	0.63	0.59	0.59
Decision Tree	1	0.68	0.77	0.72
	2	0.41	0.59	0.49
	3	0.87	0.74	0.80
	Weighted average	0.76	0.74	0.80
SVM	1	0.46	0.76	0.57
	2	0.12	0.35	0.18
	3	0.87	0.28	0.43
	Weighted average	0.66	0.46	0.46
Gradient Boosting	1	0.70	0.75	0.73
	2	0.41	0.54	0.46
	3	0.86	0.78	0.81
	Weighted average	0.77	0.75	0.75
LightGBM	1	0.76	0.80	0.78
	2	0.47	0.59	0.52
	3	0.86	0.80	0.83
	Weighted average	0.79	0.78	0.79
RF	1	0.74	0.76	0.75
	2	0.48	0.61	0.54
	3	0.86	0.80	0.83
	Weighted average	0.78	0.77	0.78
XGBoost	1	0.69	0.76	0.72
	2	0.38	0.59	0.46
	3	0.86	0.74	0.80
	Weighted average	0.76	0.73	0.74
DNN	1	0.57	0.78	0.66
	2	0.26	0.17	0.20
	3	0.78	0.64	0.71
	Weighted average	0.66	0.65	0.65
FCNN	1	0.59	0.71	0.64
	2	0.31	0.26	0.29
	3	0.76	0.68	0.72
	Weighted average	0.66	0.66	0.66

Among the deep learning models, FCNN outperformed DNN in most metrics. FCNN achieved F1-scores of 0.64 for Low Success, 0.29 for Medium Success, and 0.72 for High Success, indicating its ability to capture patterns in the data effectively

for Low and High Success but continuing to struggle with Medium Success (Table 9). DNN, in contrast, achieved its best F1-score of 0.71 for High Success but performed poorly for Medium Success (F1-score of 0.20) and Low Success (F1-score of 0.66), indicating less consistency compared to FCNN (Table 9).

The results for the public dataset align with those from the company dataset, showing once again that Random Forest, LightGBM, and XGBoost are the most reliable and consistent models for predicting sprint success levels. These models performed very well for both Low and High Success classes, often achieving F1-scores above 0.80. Similarly, FCNN demonstrated solid results, particularly for Low and High Success, but struggled with Medium Success like in the company dataset.

Evaluating the accuracy of machine learning models is essential to understand their ability to generalize and make correct predictions on data. Accuracy provides a straightforward metric for assessing how well a model performs, but it is not sufficient on its own. Variability in accuracy across different cross-validation folds can indicate how consistent and reliable a model is when faced with different data splits. Therefore, we not only need to assess the average accuracy but also examine the distribution of accuracy scores. For this purpose, we use a boxplot, which provides a detailed visualization of these distributions (Figure 11).
The boxplot shows the cross-validation accuracy scores for different machine learning models. Each box represents the interquartile range (IQR), which contains the middle 50% of accuracy scores for a model, while the whiskers extend to the minimum and maximum scores. The line inside the box indicates the median accuracy, giving insight into the central performance tendency of the model.

From the data, Random Forest and SVM exhibit relatively high median accuracy and consistent performance, making them strong candidates for reliable

predictions. Gradient Boosting, XGBoost, and LightGBM also perform well, with moderate accuracy and narrow variability, showing their robustness. On the other hand, Logistic Regression, DNN, and FCNN show lower accuracy scores with wider variability, suggesting less stable performance and higher sensitivity to data splits.

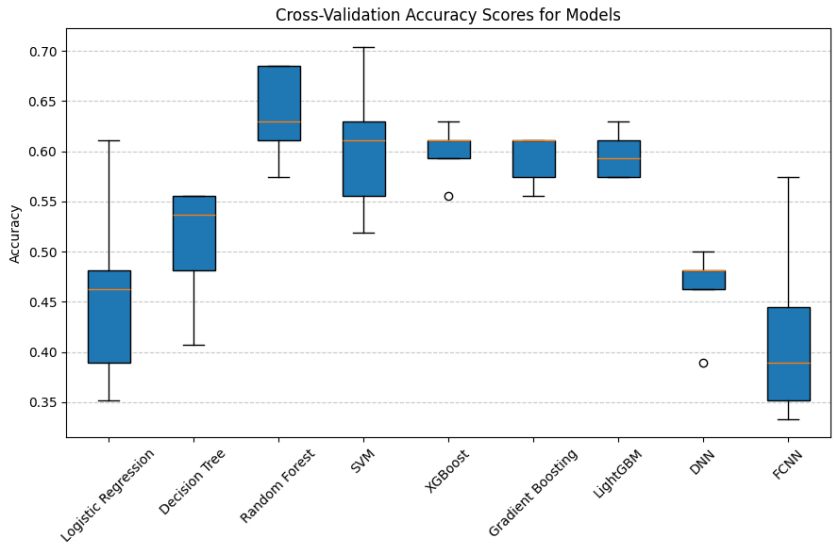

Figure 11. Comparison of model accuracy and stability

This boxplot graphic helps to see how well the models perform and understand how consistent they are. Models with smaller boxes and whiskers (e.g., LightGBM, Gradient Boosting) show greater stability, while models with larger ranges (e.g., Logistic Regression, FCNN) highlight potential inconsistencies.

4.5 Explainability

When building machine learning and deep learning models for decision-making, explainability is an important aspect of any data-driven analysis. The complexity while achieving high performance, it might be difficult to understand how and why

these predictions are made. By explainability, workings of the models and the impact of different features on the predictions can be understood.

In this study, both global and local explainability methods are explored. Feature Importance analyses are used to provide a view of which features have the most influence on the model's overall performance. Besides that, SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations) are used to go deeper, offering both global and local perspectives. SHAP, for instance, assigns a contribution value to each feature for every prediction, enabling a detailed understanding of the model's behavior across the dataset. On the other hand, LIME focuses on local explainability, analyzing individual predictions to show how specific features influenced the model's output for a particular instance. By combining global insights with detailed local explanations, this part provides a comprehensive understanding of the model's decision-making process, ensuring transparency and accountability.

4.5.1 Explainability results with model embedded feature importance technic

The bar chart displays the feature importance scores, highlighting the relative contribution of each feature in predicting the success level of sprint (Figure 12). Sufficient Description Item Percentage emerges as the most important feature, suggesting that the completeness of task descriptions significantly influences the outcome. This highlights the importance of well-documented items in ensuring clarity and producing effective results within iterations.

Right behind it are Effort M (medium effort) and Number of Items, which highlight how much the size and complexity of tasks, along with the overall workload, influence success. If tasks are balanced and manageable, sprints are more

likely to go well. Bug Percentage and Effort L (large effort) also play a role, showing that handling bugs and tackling larger tasks are important, though not as critical as the top factors.

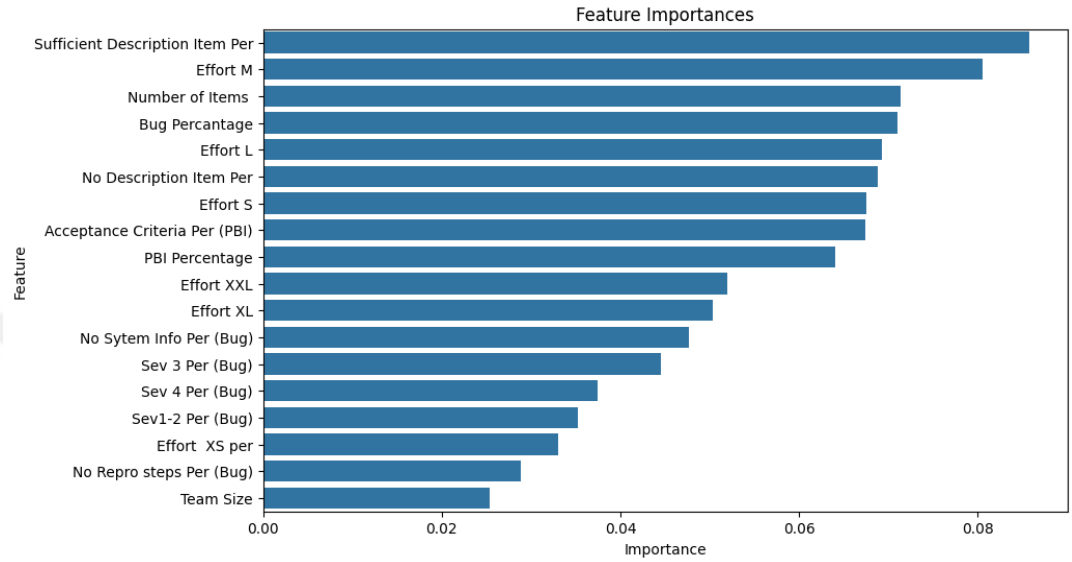

Figure 12. Feature importance for company data

On the other hand, things like Team Size, No Repro Steps Per (Bug), and Effort XS Per don't seem to have as big of an impact. While they still matter, they're not as influential as having clear tasks and a balanced workload.

In the end, the chart makes one thing very clear: success comes from planning tasks well and keeping the workload balanced. When teams focus on clarity and manage tasks effectively, great results are much more likely to follow.

Figure 13 highlights the importance of various features in public data for determining sprint success. At the top is Sprint duration, which is the most influential factor for sprint success. This shows how important it is to set the right timeline for a

sprint. A well-planned sprint duration provides the structure and speed needed for the team to stay productive and meet their goals.

The next most important factors are Number of issues at start time and Scrum team members, showing that both the initial workload and the size of the team play key roles in success. Starting with the right amount of work ensures the sprint is neither overwhelming nor underutilized, while having a balanced team size optimizes collaboration and output.

Number of to-do issues and Number of in-progress issues follow in importance, highlighting the significance of effective task management during the sprint. Tracking what's left to do and monitoring progress are essential for keeping the team focused and ensuring the sprint stays on track.

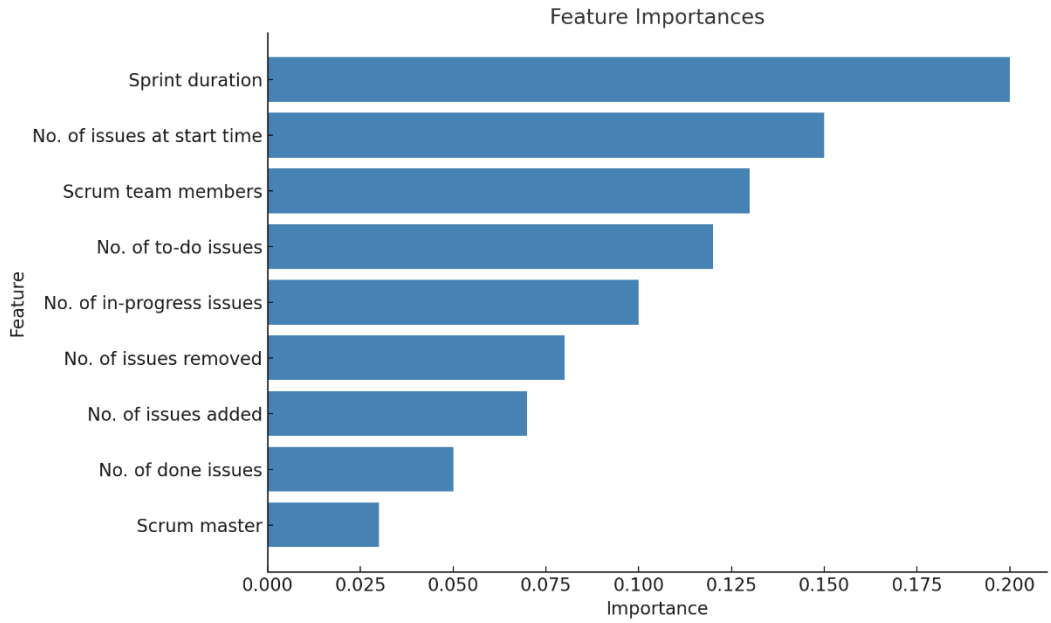

Figure 13. Feature importance for public data

Lower down the list, Number of issues removed, Number of issues added, and Number of done issues have moderate importance. These features reflect

changes made during the sprint and the progress achieved but don’t have as much impact as the initial setup and ongoing task management. Finally, Scrum master has the least influence, which isn’t surprising since the role of Scrum master in most teams is typically fixed at one person, as outlined in the Scrum guide. This consistency means the number of Scrum masters doesn’t significantly affect sprint outcomes.

In summary, this chart emphasizes that a well-balanced sprint with the right duration, workload, and team size, combined with effective task management, is the foundation for success. In-sprint adjustments play supporting roles but are secondary to the overall planning and execution of Scrum framework.

4.5.2 Explainability results with SHAP

To provide a detailed look into how different features influence the predictions made by the model, SHAP analysis is used (Figure 14). It essentially breaks down the "decision-making process" of the model, showing which factors are most important and how changes in these factors affect the output.

The y-axis lists all the features in the dataset, ordered by their overall impact on the model’s predictions. The x-axis represents the SHAP value, which indicates the direction and magnitude of the feature’s impact on the model output. Positive SHAP values (on the right) suggest that high values of the feature increase the likelihood of sprint success, while negative SHAP values (on the left) imply that high values of the feature are associated with a lower likelihood of success. The colors of the dots represent the feature values, with red indicating high values and blue

representing low values. If the red values are on the right side, it means that high values of that variable positively contribute to the sprint's success.

For descriptive features, "PBI Percentage" and "Sufficient Description Item Per" show a strong positive correlation with sprint success. When these values are high, the sprint is more likely to succeed, as indicated by the concentration of red dots on the positive SHAP side. This highlights the importance of well-defined backlog items and sufficient descriptions in ensuring sprint success. On the other hand, "Acceptance Criteria Per (PBI)" has the opposite effect—higher values negatively impact success, suggesting that overly stringent acceptance criteria might hinder progress.

Among the effort-related features, "Effort XXL" and "Effort XL" have predominantly positive SHAP values for higher feature values (red dots), indicating that larger tasks positively contribute to sprint success. Conversely, smaller tasks, such as those represented by "Effort XS" and "Effort S," have less consistent effects, with their impact being more balanced across both sides of the SHAP axis. This suggests that smaller tasks do not strongly influence sprint success and may have a neutral effect overall.

In terms of severity levels for bugs, the "Sev 1-2 Per (Bug)" feature, which represents the percentage of critical bugs, has a negative impact on sprint success as its SHAP values are mostly negative for higher feature values. Similarly, "Sev 3 Per (Bug)" shows a slight tendency towards a negative impact, though its influence appears to be less pronounced. In contrast, "Sev 4 Per (Bug)" has a more positive impact on sprint success when it has higher values, implying that lower-priority bugs are easier to address and do not hinder the sprint’s progress.

Overall, SHAP results show what the model gives importance when making predictions. The analysis highlights that success is primarily driven by well-defined backlog items, sufficient descriptions, and larger task efforts. Conversely, a higher percentage of critical bugs and overly detailed acceptance criteria negatively impact sprint outcomes. These insights can guide teams to focus on prioritizing detailed and manageable backlog items while addressing critical bugs efficiently to improve sprint success.

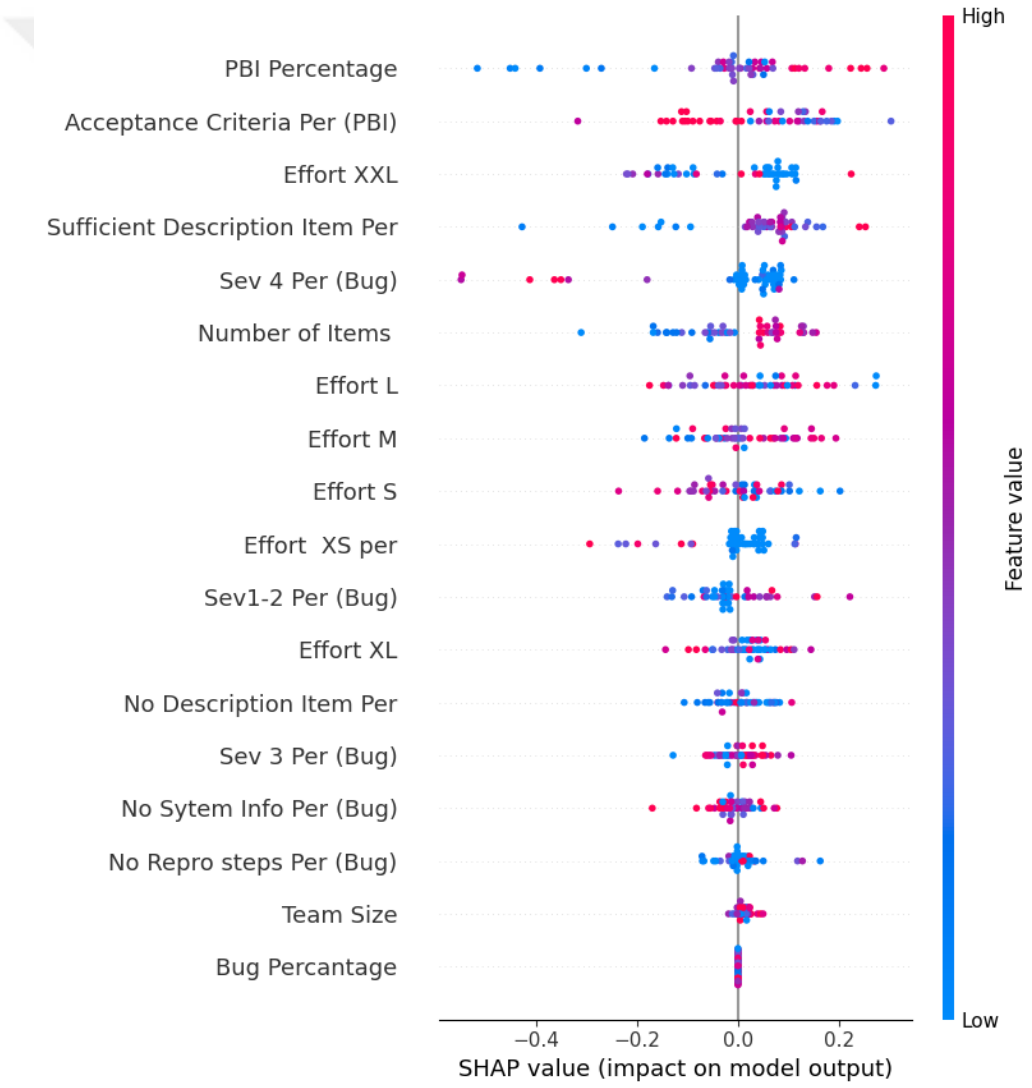

Figure 14. SHAP analysis for company data

The SHAP summary plot provides a big-picture view of which features are the most influential across the entire dataset. However, to complement this global perspective, a local analysis was performed using a SHAP force plot, focusing on a specific data point (Figure 15). This helps to see how those same features impact the prediction for that particular case.

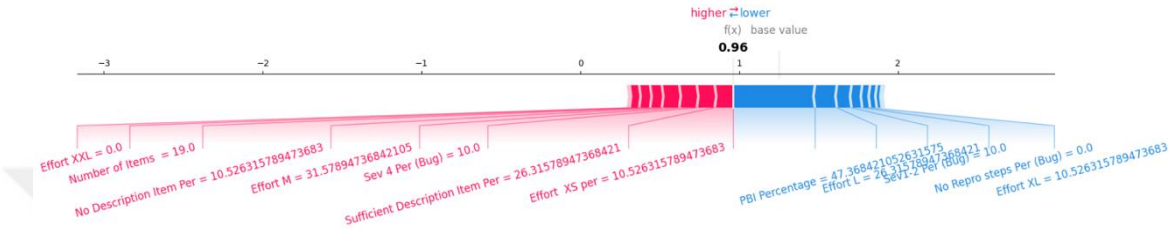

Figure 15. SHAP force plot for company data

The local SHAP plot shows how specific features impacted the success of this particular sprint. In this case, the absence of large tasks (Effort XXL = 0.0) had a strong negative effect, highlighting the importance of including larger tasks for better outcomes. A moderately high PBI Percentage (47.37) positively influenced the sprint, showing that having a good proportion of well-defined backlog items is important. Similarly, Effort XL and Effort L (medium and large tasks) contributed positively, suggesting that appropriately sized tasks support sprint success. On the other hand, No Description Item Per (31.57) and Number of Items (19.0) had negative impacts, indicating that having too many items or insufficient descriptions can make it harder to achieve success (Figure 15).

The reason for conducting the local analysis is to verify whether the global trends observed in the summary plot align with the impact of features for a particular prediction. When comparing this to the global SHAP summary plot, the local trends

mostly align with the overall patterns seen across all sprints. Globally, Effort XXL is consistently shown as a positive contributor to success, and in this local instance, its absence negatively affected the sprint. Similarly, higher PBI Percentage values are associated with success both locally and globally. Effort XL and Effort L also have positive contributions in both analyses, reinforcing the importance of medium-to-large tasks. However, features like No Description Item Per and Number of Items show more variability in the global analysis, where their impact depends on the specific sprint.

In summary, the local and global SHAP analyses mostly agree, especially on the positive effects of large tasks and well-defined backlog items. However, some features, like missing descriptions and item count, can have varying effects depending on the sprint. This highlights the value of using both local and global analyses to fully understand what drives sprint success. Together, they create a more complete, transparent, and trustworthy picture of how the model makes decisions.

The SHAP analysis conducted on the public dataset also show us a few important points influencing sprint success (Figure 16). The results show that no_issue_starttime, no_issuetodo, and planday are the most significant factors in determining sprint outcomes. In contrast, features like no_admin (number of scrum masters) and no_issueinprogress have little impact, as their SHAP values are close to zero for most cases. For no_issue_starttime, higher values are strongly linked to High Success, indicated by the red points clustering on the right side of the graph. This suggests that teams that avoid last-minute task additions or delays in starting planned work are more likely to achieve successful sprint outcomes. It underscores the importance of proper planning and adhering to the sprint schedule for better

performance. Similarly, no_issuetodo shows mixed effects, but higher values are generally associated with High Success, while lower values tend to correlate with Low Success. Planday, representing sprint duration, has a more balanced influence. Both very short and very long sprint durations occasionally lead to Low Success, showing that extreme sprint lengths might negatively impact outcomes.

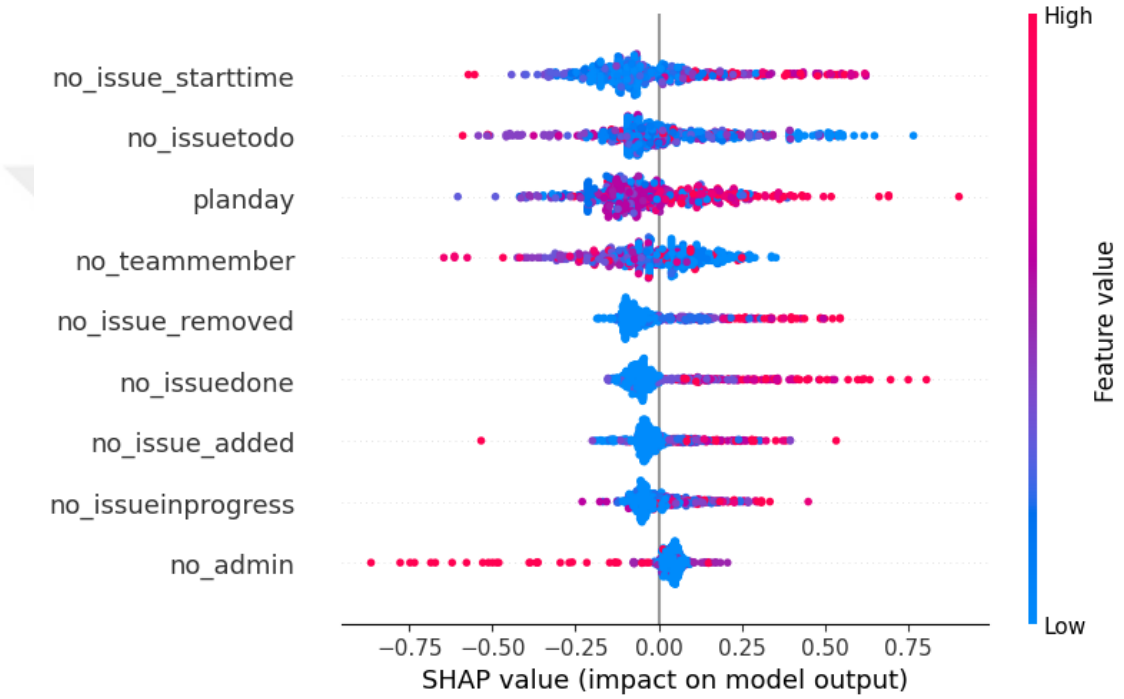

Figure 16. SHAP analysis for public data

Additionally, the analysis reveals that some features, like no_admin (number of scrum masters), are slightly more predictive of Low Success, as shown by the blue points clustering on the left. Meanwhile, features such as no_issue_removed and no_issuedone have a more balanced influence, contributing to both High and Low Success in different contexts. This analysis highlights the importance of effective task management, planning, and team dynamics in determining sprint outcomes.

Also a local analysis with using SHAP force plot is done for public data whether to see same features impact the prediction for an instance (Figure 17). The overall prediction value is 0.84, derived from the base value adjusted by the feature contributions. Features such as no_admin = 1.0 and no_teammember = 1.0 increase the prediction (shown in red), having a positive impact and this aligns with the global analysis, where these features generally show positive SHAP values across the dataset (Figure 17). Similarly, no_issue_starttime and no_issuetodo negatively influence the prediction in the local analysis (blue), and the global SHAP summary plot also shows that these features typically have negative effects, particularly when their values are low. planday consistently exhibits a negative impact in both analyses, indicating that longer planning durations reduce the prediction value. Features like no_issue_removed and no_issue_added also display matching patterns, with their contributions reducing the prediction in both local and global contexts.

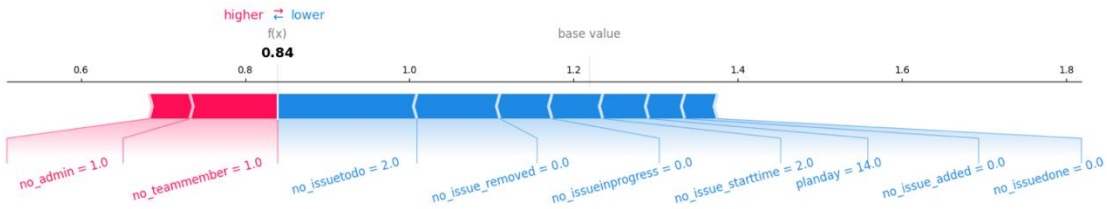

Figure 17. SHAP force plot for public data

In conclusion, the local and global SHAP analyses are largely consistent. Both highlight the negative impacts of features such as no_admin and no_teammember, as well as the positive contributions of no_issuetodo, no_issue_removed, and no_issueinprogress. This alignment demonstrates that the global trends provide a reliable summary of feature importance, while the local

analysis offers additional detail about how these features interact in specific cases. Together, they provide a comprehensive understanding of what drives sprint success.

4.5.3 Explainability results with LIME analysis

LIME analysis provides an insightful and comprehensive analysis regarding features that have contributed to the model's prediction of 0.44 for the case in Figure 18. The prediction is reasonable within the model's boundary of – 0.38 as the minimum and 2.65 as the maximum. The chart determines which features, explained in orange, contributed to the prediction by helping raise it and which, explained in blue, lowered that prediction and hence had a negative impact.

On the other hand, the Number of Items which is more than 25 in most cases was the most important, raising the prediction by 0.14. No severe bugs, Sev 4 Per (Bug) <= 0.00, also added 0.12 to the prediction proving that having no critical bugs strongly enhances the expected outcome. Having low effort tasks (Effort S <= 7.69) as well as having a sufficient description item ratio (Sufficient Description Item Per > 33.33) was also better for enhancing the expected value.

One negative contributor is the relatively lower PBI Percentage (57.14, which is lesser than 60.75), as its effect was the largest in reducing the prediction by 0.24. This shows that it is expected that for better outcomes PBI percentages should be high, and since they are lower than this threshold, the prediction would be lower. In the same way, Effort M and Effort XXL contributed negatively although their contributions were small, with larger Efforts XXL seeming to introduce some level of opposing forces or inefficiencies.

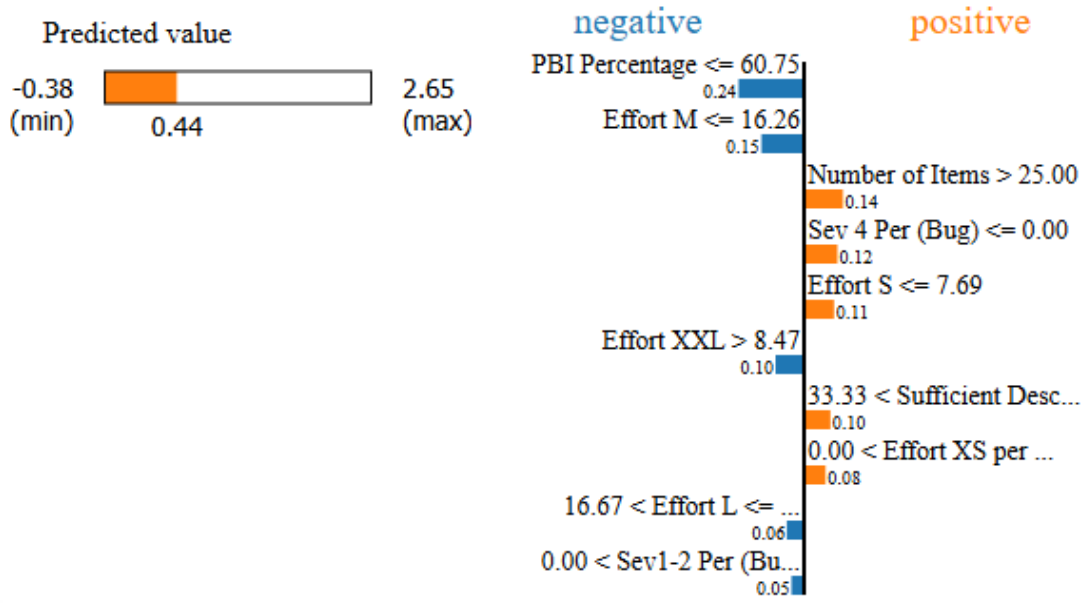

Figure 18. LIME analysis for company data

To summarize, the high number of items, absence of critical bugs and reasonable amount of description (lacking brevity) were positive constituents raising the prediction, while other factors such as low PBI and some metrics regarding the effort made an insignificant negative impact. This local explanation also provides clarity about the reasons behind the model's prediction and the contribution of each feature towards the prediction.

The LIME analysis is also done for public data to provide an instance-specific explanation of how different features influenced the model's prediction for a single sprint (Figure 19). In this case, the model's predicted value was 0.84, which leans slightly toward a "High Success" prediction. The features are divided into two categories: positive contributors (which push the prediction higher) and negative contributors (which push the prediction lower).

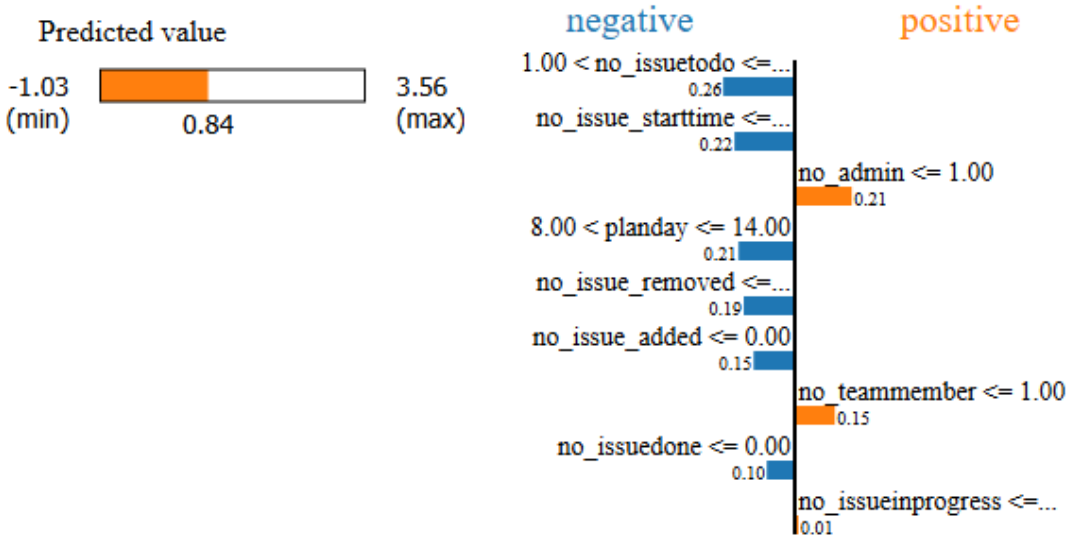

Figure 19. LIME analysis for public data

Among the positive contributors, the most influential factor is no_admin, which reflects the number of scrum master in the team. Having fewer scrum master role contributed positively to the sprint's success prediction, suggesting that smaller, more agile leadership structures may lead to better outcomes. Additionally, features like no_teammember and no_issueinprogress also pushed the prediction upward. These results imply that smaller team sizes and fewer tasks actively in progress during the sprint help improve sprint performance. On the other hand, the most impactful negative contributor is no_issuetodo, representing the number of unresolved tasks at the sprint's conclusion. A higher number of unresolved tasks lowered the prediction, highlighting the importance of completing planned work to ensure a successful sprint. Similarly, features like no_issue_starttime, which indicates delays in starting issues, and planday, the sprint duration, also negatively influenced the prediction. Longer sprint durations and delays in addressing tasks appear to reduce the likelihood of a successful sprint outcome.

CHAPTER 5

DISCUSSION

This study provides simple and clear insights into what makes sprints successful in Agile software development by addressing key questions about how machine learning (ML) can predict success, which factors are most influential, and how these insights can improve planning and decision-making.

RQ1: How effective are machine learning models in predicting sprint success in Agile software projects?

The results show that machine learning models are very effective in predicting sprint success. The models used in this study were able to analyze complex patterns in sprint data and provide accurate predictions of success or failure. For example, the models could distinguish between successful and unsuccessful sprints based on task details, workload balance, and bug data. This means that ML can serve as a valuable tool for teams looking to identify potential risks and opportunities early in the sprint process. What makes this even more practical is the use of explainability tools like SHAP and LIME, which made the predictions understandable for team members. By showing how different factors influence the predictions, these tools ensure that teams can trust and act on the results.

RQ2: What are the key factors influencing sprint success, and can explainability techniques in machine learning provide actionable insights into these factors?

Several key factors were found to influence sprint success, and explainability techniques in machine learning provided actionable insights into these factors. By analyzing features related to task clarity, workload balance, and bug management,

the study demonstrated how data-driven approaches could help Agile teams improve their planning and execution processes. The explainability tools, SHAP and LIME, not only highlighted the importance of these factors but also provided detailed, interpretable insights that teams could use to make informed decisions.

Task clarity emerged as one of the most critical elements in predicting sprint success. One of the most important findings is that having sufficient explanations for items was the top factor in predicting success. This means that having tasks with clear and detailed descriptions makes it easier for teams to stay on track and achieve their goals. When backlog items were clearly documented, teams had a better understanding of what was required, reducing confusion and ensuring alignment. For example, detailed descriptions allowed team members to collaborate more effectively, ultimately improving the sprint's overall performance.

Another significant factor was workload balance. Metrics such as the number of items planned for the sprint and effort-based categorizations, like medium and large-sized tasks, demonstrated that maintaining a balanced workload was essential. Overloaded sprints often led to missed goals and increased stress on team members, while underloaded sprints resulted in wasted resources and inefficiencies. Striking the right balance ensured that the team could manage their tasks effectively, leading to higher success rates.

Bug management, while less influential than task clarity and workload balance, still played a role in sprint success. Metrics such as the percentage of bug-related tasks and the severity of critical bugs highlighted the importance of quality management during sprints. A high proportion of critical bugs negatively affected sprint outcomes, emphasizing the need to address these issues proactively during

planning. Reducing the presence of severe bugs allowed teams to maintain a smoother workflow and focus on delivering value.

The public dataset highlighted different but related factors. Sprint Duration was the most important, showing that longer sprints and delays in tackling tasks seem to lower the chances of having a successful sprint. Other factors, like the count of planned items at the beginning of the sprint and the number of scrum team members, showed that starting with the right amount of work and having a well-sized team improve collaboration and efficiency.

Explainability techniques like SHAP and LIME provided critical insights into how these factors influenced sprint success. SHAP analysis quantified the contribution of each feature, showing, for example, that a higher percentage of sufficiently described tasks strongly correlated with successful sprints. This level of understanding helped teams prioritize key aspects of sprint planning, such as improving task documentation. LIME offered case-specific explanations, illustrating how individual features affected specific sprint predictions. For instance, having clear tasks and avoiding large, unmanageable efforts helped improve outcomes. These tools bridged the gap between complex machine learning models and practical, actionable insights, empowering Agile teams to refine their processes effectively.

Overall, using sprint success predictions enables teams to make data-driven decisions, improving their confidence in planning strategies. These insights allow teams to continuously adapt and improve their workflows, ultimately increasing the likelihood of achieving their sprint goals. This approach not only helps teams work more efficiently but also fosters a culture of continuous improvement in Agile software development.

In comparison to existing literature, this research shifts the focus from general Agile principles or project-level planning to measurable, sprint-specific factors. Previous studies often lacked measurable guidance for improving sprint outcomes, whereas this study identifies practical elements such as team skills, task clarity, and sufficient backlog allocation as key to sprint success. By aligning these metrics with machine learning predictions, this study offers concrete guidance that Agile teams can directly apply to improve sprint outcomes.

CHAPTER 6

CONCLUSION

This study explored the factors influencing sprint success in Agile software development and the role of machine learning (ML) and explainability techniques in enhancing sprint planning and decision-making processes. The findings reveal that data-driven approaches, powered by ML models, can significantly improve the accuracy of sprint planning compared to traditional estimation methods. Factors such as well-defined sprint items and sufficient inclusion of backlog tasks were identified as critical to sprint success. By integrating SHAP and LIME explainability techniques, the "black-box" nature of ML models was addressed, providing teams with actionable insights into the drivers of predictions and fostering trust in model outputs.

6.1 Practical Implications

The ability to predict sprint success has a significant impact on improving decision-making and planning strategies in Agile software projects. By leveraging these predictions, teams can focus on key areas that directly affect the success of their sprints and make informed adjustments to their processes.

One of the most important benefits is the ability to refine planning processes. Predictions highlight the critical role of task clarity, showing that well-documented and detailed backlog items are essential for success. With this knowledge, teams can prioritize creating tasks with clear descriptions and acceptance criteria during planning sessions. This reduces ambiguity and ensures that everyone on the team

understands the goals and expectations, leading to smoother execution during the sprint.

Workload balance is another area where predictions prove valuable. By analyzing the workload distribution, teams can allocate tasks more effectively, ensuring that no single sprint is overloaded or underutilized. Overloading a sprint often leads to burnout and missed deadlines, while underutilization wastes valuable resources and reduces team productivity. With predictive insights, teams can find the right balance, making sure that the workload is manageable and aligned with their capacity.

Predictions also help teams address quality issues more proactively. By identifying the impact of bug-related metrics, such as the percentage of critical bugs, teams can allocate resources to resolve these issues early in the sprint. This proactive approach minimizes disruptions and ensures that the team can maintain a steady workflow. Addressing quality issues early also reduces the need for last-minute fixes, allowing teams to focus on delivering value.

6.2 Limitations and future work

Despite these advancements, the study has certain limitations. One key limitation is the imbalance in the dataset, with a smaller number of sprints classified as “low” or “medium” success. This is likely due to the nature of Scrum teams, which focus on continuous improvement, often resulting in more successful sprints. While this highlights the strength of Agile methodologies, it also reduces the availability of diverse examples in less successful categories for analysis. Additionally, while the study used real-world data, a larger and more varied dataset could enhance the generalizability of the findings and improve model robustness.

One of the main challenges encountered during data collection was standardizing the dataset. Different projects store their data in various formats and define their items using different methodologies, making it difficult to create a unified dataset. However, collecting data from a large company like Siemens provided an advantage in terms of project diversity. To mitigate the standardization challenge, we specifically selected projects that maintained their data in a similar structure, ensuring consistency in the dataset used for this study.

Looking ahead, there are several directions for future research. Expanding the dataset to include more examples from different teams, industries, and contexts would allow for a deeper understanding of the factors influencing sprint success. This could help address the current limitations caused by the imbalance in success categories. Moreover, future studies could place greater emphasis on team competencies, exploring how skill levels, collaboration, and communication dynamics impact sprint performance. By integrating these additional dimensions, future research can provide even more comprehensive insights into improving Agile workflows.

In conclusion, this study demonstrates the value of combining Agile principles with machine learning and explainability techniques to optimize sprint success. While there are opportunities to build on these findings, the research provides a strong foundation for making data-driven improvements in Agile software development.

REFERENCES

Abadeer, M., & Sabetzadeh, M. (2021). Machine learning-based estimation of story points in Agile development: Industrial experience and lessons learned. *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, 106–115. https://doi.org/10.1109/REW53955.2021.00022

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). Manifesto for Agile Software Development. *Agile Alliance*. https://agilemanifesto.org/

Breiman, L., Friedman, J., Olshen, R.A., & Stone, C.J. (1984). *Classification and Regression Trees (1st ed.)*. Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

Breiman, L. Random Forests. *Machine Learning, 45*, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16*), 785-794. https://doi.org/10.1145/2939672.2939785

Choetkiertikul, M., Dam, H. K., Tran, T., Ghose, A., & Grundy, J. (2018). Predicting delivery capability in iterative software development. *IEEE Transactions on Software Engineering, 44*(6), 551–569. https://doi.org/10.1109/TSE.2017.2693989

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software, 81*(6), 961–971. https://doi.org/10.1016/j.jss.2007.08.020

Cohn, M. (2005). *Agile estimating and planning*. Upper Saddle River, NJ: Pearson Education.

Cortes, C., Vapnik, V. (1995). Support-vector networks. *Machine Learning 20*, 273–297. https://doi.org/10.1007/BF00994018

De Mast, J., & Kemper, B. P. H. (2009). Principles of exploratory data analysis in problem solving: What can we learn from a well-known case? *Quality Engineering, 21*(4), 366–375. https://doi.org/10.1080/08982110903188276

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv*. https://doi.org/10.48550/arXiv.1702.08608.

Drury-Grogan, M. L. (2014). Performance on agile teams: Relating iteration objectives and critical decisions to project management success factors. *Information and Software Technology, 56*(4), 506-515. https://doi.org/10.1016/j.infsof.2013.11.003

Ghimire, D., & Charters, S. (2022). The impact of Agile development practices on project outcomes. *Software, 1*(3), 265-275. https://doi.org/10.3390/software1030012

Goutte, C., & Gaussier, E. (2005). A probabilistic interpretation of precision, recall, and F-score, with implications for evaluation. *Proceedings of the European Colloquium on IR Research (ECIR'05)*, 345–359. Springer. https://doi.org/10.1007/978-3-540-31865-1_25

Fang, C., & Marle, F. (2012). A simulation-based risk network model for decision support in project risk management. *Decision Support Systems, 52*(3), 635–644. https://doi.org/10.1016/j.dss.2011.10.021

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics, 29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression (3rd ed.)*. Wiley.

Hu, Y., Zhang, X., Ngai, E., Cai, R., & Liu, M. (2013). Software project risk analysis using Bayesian networks with causality constraints. *Decision Support Systems, 56*, 439–449. https://doi.org/10.1016/j.dss.2012.11.001

Huang, L., Port, D., Wang, L., Xie, T., & Menzies, T. (2010). Text mining in supporting software systems risk assurance. *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 163–167. https://doi.org/10.1145/1858996.1859025

Kavitha, M., Gnaneswar, G., Dinesh, R., Rohith Sai, Y., & Sai Suraj, R. (2021). Heart Disease Prediction using Hybrid Machine Learning Model. *Proceedings of the Sixth International Conference on Inventive Computation Technologies (ICICT 2021). IEEE*. https://doi.org/10.1109/ICICT50816.2021.9358597

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems, 30*.

Kocaguneli, E., Menzies, T., & Keung, J. W. (2012). On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering, 38*(6), 1403–1416. https://doi.org/10.1109/TSE.2012.21

Kuhrmann, M., Tell, P., Hebig, R., et al. (2022). What makes Agile software development Agile? *IEEE Transactions on Software Engineering, 48*(9), 3523-3535. https://doi.org/10.1109/TSE.2021.3099999

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444. https://doi.org/10.1038/nature14539

Li, T., & Han, L. (2023). Dealing with explainability requirements for machine learning systems. *Proceedings of the 47th IEEE Annual Computers, Software, and Applications Conference (COMPSAC)*, 1203–1208. https://doi.org/10.1109/COMPSAC57700.2023.00182

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, Long Beach, CA. https://doi.org/10.48550/arXiv.1705.07874

Meiliana, Daniella, G., Wijaya, N., Putra, N. G. E., & Efata, R. (2023). Agile software development effort estimation based on product backlog items. *Procedia Computer Science, 227*, 186–193. https://doi.org/10.1016/j.procs.2023.10.516

Microsoft. (n.d.). Manage bugs. Retrieved December 9, 2024, from https://learn.microsoft.com/en-us/azure/devops/boards/backlogs/manage-bugs?view=azure-devops#bug-work-item-type

Mienye, I. D., & Jere, N. (2024). A survey of decision trees: Concepts, algorithms, and applications. *IEEE Access*, *12*, 86716-86727. https://doi.org/10.1109/ACCESS.2024.3416838

Mockus, A., Weiss, D. M., & Zhang, P. (2003). Understanding and predicting effort in software projects. *Proceedings of the 25th International Conference on Software Engineering*, 274–284. https://doi.org/10.1109/ICSE.2003.1201207

Moreno-García, M. N., Roman, I. R., García-Peñalvo, F. J., & Bonilla, M. T. (2008). An association rule mining method for estimating the impact of project management policies on software quality, development time and effort. *Expert Systems with Applications, 34*(1), 522–529. https://doi.org/10.1016/j.eswa.2006.09.022

Nevendra, M., & Singh, P. (2022). Empirical investigation of hyperparameter optimization for software defect count prediction. *Expert Systems with Applications, 191*, 116217. https://doi.org/10.1016/j.eswa.2021.116217

Ozcelikkan, N., Tuzkaya, G., Alabas-Uslu, C., & Sennaroglu, B. (2022). A multi-objective agile project planning model and a comparative meta-heuristic approach. *Information and Software Technology*, *151*, 107023. https://doi.org/10.1016/j.infsof.2022.107023

Hoda, R., and Noble, J. (2017) Becoming Agile: A Grounded Theory of Agile Transitions in Practice, *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE*), Buenos Aires, Argentina, 141-151, https://doi: 10.1109/ICSE.2017.21

Ranawana, R., & Karunananda, A. S. (2021). An agile software development life cycle model for machine learning application development. *2021 5th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI)*, 1–6. https://doi.org/10.1109/SLAAI-ICAI54477.2021.9664736

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144. https://doi.org/10.1145/2939672.2939778

Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process.* Addison-Wesley.

Salido, M. G., Borrego, G., Palacio Cinco, R. R., & Rodríguez, L.-F. (2023). Agile software engineers' affective states, their performance and software quality: A systematic mapping review. *Journal of Systems & Software, 204*, 111800. https://doi.org/10.1016/j.jss.2023.111800

Sarro, F., Petrozziello, A., & Harman, M. (2016). Multi-objective software effort estimation. *Proceedings of the 38th International Conference on Software Engineering*, 619–630. https://doi.org/10.1145/2884781.2884830

Scrum.org. (n.d.). Practical Fibonacci: A Beginner's Guide to Relative Sizing. Retrieved December 9, 2024, from https://www.scrum.org/resources/blog/practical-fibonacci-beginners-guide-relative-sizing

Schwaber, K. (1996). SCRUM development process. *Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*. Retrieved from https://www.scrum.org

Simon, S., Kolyada, N., Akiki, C., Potthast, M., Stein, B., & Siegmund, N. (2023). Exploring hyperparameter usage and tuning in machine learning research. *Proceedings of the IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*. https://doi.org/10.1109/CAIN58948.2023.00016

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management, 45*(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

Zafar, M. R., & Khan, N. (2021). Deterministic Local Interpretable Model-Agnostic Explanations for Stable Explainability. *Machine Learning and Knowledge Extraction, 3*(3), 525–541. https://doi.org/10.3390/make3030027