Modeling Effort Estimation and Planning in Large-Scale Agile Software Development

Kula, E.

DOI
10.4233/uuid:bac03d30-f65e-49f9-9feb-aae8f67122b6

Publication date
2025

Document Version
Final published version

Citation (APA)
Kula, E. (2025). *Modeling Effort Estimation and Planning in Large-Scale Agile Software Development*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:bac03d30-f65e-49f9-9feb-aae8f67122b6

Modeling Effort Estimation and Planning in Large-Scale Agile Software Development

Elvan Kula

Modeling Effort Estimation and Planning in Large-Scale Agile Software Development

Modeling Effort Estimation and Planning in Large-Scale Agile Software Development

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof. dr. ir. T.H.J.J. van der Hagen,
Chair of the Board for Doctorates
to be defended publicly on
Thursday, 10 April 2025, at 15:00,

by

Elvan KULA

Master of Science in Computer Science,
Delft University of Technology, The Netherlands,
born in Winschoten, The Netherlands.

This dissertation has been approved by:

Prof. dr. A. van Deursen	promotor
Dr. ir. G. Gousios	copromotor

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof. dr. A. van Deursen	Delft University of Technology
Dr. ir. G. Gousios	Delft University of Technology
Independent Members:	
Prof. dr. ir. N. Bharosa	Delft University of Technology
Prof. dr. R. Feldt	Chalmers University of Technology, Sweden
Prof. dr. ir. A. Iosup	Vrije Universiteit Amsterdam, The Netherlands
Prof. dr. M. Jørgensen	Oslo Metropolitan University, Norway
Dr. P. Thongtanunam	University of Melbourne, Australia
Prof. dr. ir. D. Spinellis	Delft University of Technology, reserve member

The work presented in this thesis has been conducted as part of the AI for Fintech Research Lab at ING, operating under the Innovation Center for Artificial Intelligence (ICAI) flag.

Keywords: Software Effort Estimation, Sprint Planning, Agile Processes

Printed by: ProefschriftMaken, `www.proefschriftmaken.nl`

Cover: Nathan Dumlao

Style: TU Delft House Style, with modifications by Moritz Beller
`https://github.com/Inventitech/phd-thesis-template`

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.

ISBN 978-94-6510-552-9

An electronic version of this dissertation is available at
`https://repository.tudelft.nl/`.

Somewhere, something incredible is waiting to be known.

Carl Sagan

Contents

Summary

Late deliveries have been a common problem in the software industry for decades. They often result from deficiencies in effort estimation and project planning. These deficiencies arise due to the complexity of software development, where various social and technical factors affect project effort and scheduling. Variability in human elements, such as team dynamics and changing user requirements, adds further uncertainty. Since meeting time and cost estimates is crucial for project success, improving effort estimation and planning remains a key priority for software organizations. More accurate forecasting enables better resource allocation, reduces delays, and enhances customer satisfaction.

Over the past two decades, software organizations have increasingly adopted agile methods to improve flexibility and responsiveness. However, despite these advantages, schedule delays remain common, with nearly half of agile projects experiencing overruns of 25% or more. A key challenge lies in balancing the flexible, short-term planning of small functionalities (*user stories*) with the structured, long-term planning required for larger development units (*epics*). Current industry practices offer limited support for managing these complexities, especially in large-scale agile settings.

This thesis presents a novel suite of expert- and data-based strategies to improve effort estimation and planning in large-scale agile software development. We conduct a series of case studies at ING, a large Dutch internationally operating bank, to collect and analyze data from hundreds of agile teams and projects. We identify key factors influencing delays in epics and user stories and develop models to predict delays at both levels. At the epic level, we compile our findings into a conceptual framework representing influential factors and their relationships to on-time delivery. Additionally, we explore dynamic Bayesian methods to continuously update delay predictions throughout an epic's development life cycle. At the story level, we examine how team characteristics affect the likelihood of delays. We also investigate how these factors, combined with incremental learning methods, can improve story delay predictions. Finally, we develop a model that optimizes sprint plans based on team goals and delivery performance.

Our research identifies 25 factors and their interactions that affect the on-time delivery of epics. The most influential factors are predominantly social in nature, such as task dependencies, organizational alignment, and internal politics. These factors interact hierarchically: organizational factors shape team behavior, which in turn affects technical factors. To capture these complexities, we demonstrate that dynamic Bayesian methods, using delay patterns as input, effectively update delay predictions as new information becomes available. At the story level, our findings suggest that planning in agile settings can be significantly improved by integrating team-related information and incremental learning methods into predictive models. Moreover, we find that user story prioritization depends on a combination of factors that vary by project context. Our sprint plan optimization model effectively addresses this variability and generates plans that deliver more business value, align more closely with sprint goals, and mitigate delay risks better.

Samenvatting

Late opleveringen zijn al decennialang een veelvoorkomend probleem in de software industrie. Ze zijn vaak het gevolg van onnauwkeurige inschattingen van de benodigde hoeveelheid werk en de projectplanning. Deze onnauwkeurigheden komen vooral voort uit de complexiteit van softwareontwikkeling, waarbij een breed scala aan sociale en technische factoren de benodigde inzet en planning beïnvloeden. De variabiliteit in menselijke aspecten, zoals teamdynamiek en veranderende gebruikersvereisten, brengt extra onzekerheid met zich mee. Aangezien het succes van projecten sterk afhankelijk is van het behalen van tijd- en kosteninschattingen, blijft het verbeteren van de nauwkeurigheid van werkinschattingen en planning een topprioriteit voor softwareorganisaties. Nauwkeurigere voorspellingen maken een betere toewijzing van middelen mogelijk, verminderen vertragingen en verhogen de klanttevredenheid.

De afgelopen twee decennia hebben softwareorganisaties in toenemende mate agile methoden omarmd om hun flexibiliteit en responsiviteit te verbeteren. Ondanks deze voordelen blijven projectvertragingen een veelvoorkomend probleem, waarbij bijna de helft van de agile projecten te maken krijgt met overschrijdingen van 25% of meer. Een grote uitdaging ligt in het balanceren van de flexibele, kortetermijnplanning van kleine functionaliteiten (*user stories*) met de gestructureerde, langetermijnplanning die nodig is voor grotere functionaliteiten (*epics*). De huidige ondersteuning voor softwareorganisaties op dit gebied is beperkt, vooral binnen grootschalige agile omgevingen.

Dit proefschrift presenteert een innovatieve reeks op expertise en data gebaseerde strategieën om werkinschattingen en planning in grootschalige agile softwareontwikkeling te verbeteren. We voeren een reeks case studies uit bij ING, een grote internationaal opererende Nederlandse bank, om gegevens te verzamelen en te analyseren van honderden agile teams en projecten. We identificeren de belangrijkste factoren die vertragingen beïnvloeden en ontwikkelen modellen om deze zowel op epic- als story-niveau te voorspellen. Op het niveau van epics bundelen we onze bevindingen in een conceptueel raamwerk dat invloedrijke factoren en hun verbanden met tijdige oplevering weergeeft. Daarnaast onderzoeken we dynamische Bayesiaanse methoden om continu vertragingen te voorspellen gedurende de ontwikkelingscyclus van een epic. Op het niveau van stories analyseren we hoe teamkenmerken de kans op vertragingen beïnvloeden. We onderzoeken ook hoe deze factoren, in combinatie met incrementele leermethoden, de voorspelling van vertragingen in user stories kunnen verbeteren. Tot slot ontwikkelen we een model dat sprint plannen optimaliseert op basis van de doelen en leveringsprestaties van teams.

Ons onderzoek identificeert 25 factoren en hun onderlinge interacties die van invloed zijn op de tijdige oplevering van epics. De meest invloedrijke factoren zijn overwegend sociaal van aard, zoals taakafhankelijkheden, organisatorische afstemming en interne politiek. Deze factoren interacteren hiërarchisch: organisatorische factoren beïnvloeden het teamgedrag, dat op zijn beurt de technische factoren beïnvloedt. We tonen aan dat dynamische Bayesiaanse methoden, met vertragingspatronen als invoer, deze complexiteit

effectief kunnen vastleggen en vertragingen nauwkeurig blijven bijwerken zodra nieuwe informatie beschikbaar komt. Op story-niveau suggereren onze bevindingen dat de planning in agile omgevingen aanzienlijk kan worden verbeterd door teamgerelateerde informatie en incrementele leermethoden te integreren in voorspellende modellen. Daarnaast blijkt dat de prioritering van user stories afhankelijk is van een combinatie van factoren die variëren per projectcontext. Ons model voor de optimalisatie van sprint plannen speelt effectief in op deze variabiliteit en genereert plannen die meer bedrijfswaarde opleveren, beter afgestemd zijn op sprintdoelen en vertragingen effectiever helpen te beperken.

Acknowledgments

It was September 2013 when I first stepped onto the TU Delft campus, ready to begin my bachelor's in Computer Science. I was fascinated by the people, the architecture, and the exciting research happening around me. Little did I know that I would spend the next decade there, completing my bachelor's, master's, and PhD. Along the way, I met so many incredible people, each inspiring me in their own way. I want to take a moment to express my heartfelt gratitude to everyone who has been part of my PhD journey.

Arie: From my master's thesis to my PhD, you have been there every step of the way. Thank you for giving me the chance to do research on what some might call a 'dinosaur' of a topic, but one that continues to fascinate me. I have always appreciated the energy and positive mindset you brought to our meetings, making even the toughest discussions constructive and motivating. I have learned so much from you, but if I had to highlight one lesson, it would be how to turn negative situations into positive ones – a skill that extends far beyond academia. Your feedback helped me push each paper to the next level, but more than that, you showed me that research is not just about technical work. It is just as much about stakeholder management and storytelling. I am also thankful for your guidance during my time as a lab manager at ING, helping me balance academic and business perspectives in an industry setting. Your support and the freedom you gave me to explore my own path have meant a lot. For all of that, I cannot thank you enough!

Georgios: Back in 2018, when we first discussed topics for my master's thesis, I had just met you, and I immediately noticed two things – your impressive technical expertise and our shared high level of ambition. Now, after completing both my master's thesis and PhD with you, I can say that my first impression was absolutely right. No matter how complex the challenge or how tight the deadline, your response was always a confident "Why not?". Your support, honest feedback, and ability to challenge my ideas while still giving me the space to explore my own have been truly invaluable. Thank you for being there when I needed support, for offering advice when I sought it, and for constantly pushing me to grow. I have learned a lot from you – not just about technical work, but also about life and the importance of sometimes stepping back to enjoy the little things. From research discussions to the occasional SAL group dinners, this journey has been an incredible one, and I am truly grateful to have shared it with you.

Joost: Thank you for giving me the incredible opportunity to pursue my PhD within ING. Your trust and confidence in me made this journey possible, and I am truly grateful for the chance to have worked in such a unique research setting. I also want to thank you for being such a supportive manager. I did not fully realize how much you handled behind the scenes to create a smooth and focused working environment for us until I became a manager myself. Looking back, I appreciate it even more. Your leadership style has inspired me, and I hope to carry that same approach forward in my own career.

Eric, Jerry, Serge, Evert-Jan: From day one, you welcomed me into your product team as if I had always been part of it. You were eager to create opportunities for me, setting

up meetings with management to help showcase my research. I am grateful for all the great discussions and brainstorming sessions we had – many of which ran over time, but I believe that only reflects how passionate you are about your work. Thank you for helping me find my place at ING and for making me feel at home in the team.

Hennie: Thank you for being my first buddy at ING and for helping me grow quickly as a researcher. Working with you to professionalize research at ING was both rewarding and a lot of fun. Our trip to the summer school at Elba Island was a highlight – an experience I will always look back on with a smile!

Mieke, Shiler, Kerem, Andries: Thank you for trusting me and giving me the freedom to run the AFR lab in my own way. Your confidence in me allowed me to grow both professionally and personally. I am grateful for everything I learned from you, especially how to connect academia with real-world business problems.

AFR Lab (Lorena, Eileen, Luís, Floris, George, Sara, Patrick, Arumoy, Leonhard, Kevin, Ralf, and all the master students): Oh, what a ride it has been! Starting a research lab in the midst of COVID was no small feat – but through it all, we created something truly special. Thank you for making the ING lab feel like a second home. From inspiring discussions to unforgettable laughs and countless coffee breaks, I have cherished every moment. I loved getting to know each of you and being part of your journeys. The friendships we built mean a lot to me, and our time together will always hold a special place in my heart.

Pradyot, Thomas, Joaquin, Marius, Remco: I am grateful to have supervised your research projects at ING. Working with you was a true pleasure, and in many ways, I learned as much from you as you (hopefully) did from me. Your enthusiasm and curiosity often made me see my own research in a new light. Thank you for the great collaborations. It was a privilege to be part of your academic journey!

SAL (Amir, Ayushi, Enrique, Joseph, Maliheh, Mehdi): Thank you for all the insightful discussions, the lunches that always ran longer than planned ☺, and the much-needed coffee breaks that made this PhD journey all the more enjoyable. A special shout-out to Joseph and Mali for being there to share the highs and lows of PhD life – knowing I was not alone in the struggle made all the difference!

SERG: Being part of SERG for so many years has been an incredible experience. With the group growing rapidly, it is impossible to name everyone, but I am truly grateful to all past and present members for the inspiring discussions, collaborations, and fun moments we have shared. From brainstorming research ideas to Friday beers at The Hangover, being surrounded by such a talented group made this journey even more special. Minaksie and Kim, thank you for always keeping everything running smoothly!

Alexandru, Diomidis, Magne, Nitesh, Pick, Robert: Thank you for agreeing to be part of my defense committee, for reviewing my dissertation, and for traveling to Delft. I truly appreciate the time and effort.

Carlos: Even though we met during the final stretch of my PhD, I am grateful for the great discussions we had about dissertation writing, academic life, and life in general. Thank you for being a great listener and for making that final phase a little easier!

Tim, Stefan, Otto, Pim Otte, Pim Veldhuisen: From game nights and movie nights to all the drinks we have shared at the /Pub, you became the family I found in Delft. The past few years have been busier than I would have liked, especially after moving to Almere, but I am looking forward to making even more great memories together in the future!

Monica: Darling, thank you for always being there for me, offering your support and a listening ear whenever I needed it. I am so grateful for all our travel adventures, endless coffee chats, delicious chocolate cakes, and peaceful dog walks. We have created so many wonderful memories together, and I look forward to many more!

Spike, Frodo, Sam, Benji: Thank you for keeping me company during late-night writing sessions and offering just the right (and sometimes not-so-needed) distractions. From lying on my keyboard at the worst possible moments to making me smile when I needed it most, you were the best little supporters throughout this journey! 😸

Yud, Mama, Papa: Canlarım benim, çok sağ olun her zaman yanımda olduğunuz için ve her zaman yüzümü güldürdüğünüz için. Sizi çok seviyorum ♥ *Papa*: Thank you for inspiring my love for science and engineering from an early age. You often spoke about how much you loved the idea of doing a PhD, and even before I understood what that meant, you planted the seed that led me here. You taught me that walking my own path is a good thing, to follow my heart, and to always stay true to myself. No matter where life takes me, I carry your lessons with me forever.

Last but not least, I am forever grateful to my best friend and soon-to-be husband, *Jesse:* You have been by my side through all the ups and downs, and I honestly do not know how I would have made it through without your love and support. Thank you for always lifting me up, making me laugh, and being my rock through all these years! ♥

Elvan
Almere, July 2024

1

Introduction

Late deliveries continue to be a major challenge in the software industry. Across sectors, software projects face the highest risk of schedule overruns [1], running on average approximately 30% over time [1, 2]. Despite decades of research, this percentage does not seem to have decreased since the 1980s [3]. A study by McKinsey and the University of Oxford [4] reported even more concerning figures: 17% of large-scale software projects experience severe delays, with overruns reaching an average of 200%. These issues can lead to substantial financial losses, with each additional year spent on a project increasing cost overruns by an estimated 15% [4]. As a result, project delays pose significant business risks, including missed market opportunities, customer dissatisfaction, and reduced competitiveness [5, 6].

Schedule overruns are manifestations of deficiencies in effort estimation and project planning. These deficiencies largely arise due to the complexity of software development, where various social and technical factors affect project effort and scheduling [1, 7]. Variability in human elements, such as team dynamics and changing user requirements, adds further uncertainty [8]. Since meeting time and cost estimates is crucial for project success [9], improving the accuracy of effort estimation and planning remains a key priority for software organizations. More accurate forecasting enables better resource allocation, reduces delays, and increases stakeholder confidence.

Over the past two decades, agile methodologies have become the dominant approach for managing software projects [10]. By enabling incremental development through short iterations, agile methods allow teams to adapt to changing priorities, rapidly deliver business value, and mitigate risk [11]. However, despite these advantages, predicting and managing delivery timelines remains a critical challenge. Nearly half of agile software projects experience schedule overruns of 25% or more [12].

Agile projects rely on iterative, short-term planning in which teams refine their estimates throughout the project life cycle [13]. Central to this planning is the ability to predict, at any phase of the project, whether planned software features can be delivered on-time. This is particularly challenging due to the need to integrate the flexible, short-term planning of small functionalities (expressed as *user stories* [14]) with long-term commitments to larger development units (referred to as *epics* [15]). Most agile teams rely

heavily on expert judgment for effort estimation and delivery forecasting [10, 16], which introduces subjectivity and potential inconsistencies. To improve on-time delivery, there is a need for automated approaches that can enhance delay prediction and planning at both the user story and epic levels. Such predictive models would enable software project managers and development teams to anticipate potential delays, proactively adjust their planning, and take corrective actions to keep projects on schedule.

A key challenge in developing automated support for estimation and planning is identifying the key factors that influence on-time delivery in software projects. Prior research identified a large pool of factors that may affect the software development effort [17], but their relative impact remains unclear. Moreover, we lack an understanding of the relationships between these factors and how they impact on-time delivery. Existing insights are largely based on expert judgment, with limited validation using quantitative data from project repositories. Previous studies have developed models for effort estimation at the user story level [18–20], predicting delays in bug-fixing and issue resolution [21–23], and forecasting delivery capacity at the iteration level [24, 25]. However, little work has focused on models specifically designed for delay prediction and planning in agile settings.

This thesis aims to fill these research gaps by providing insights and automated support for predicting delays and improving planning in agile software development. To achieve this, we conduct a series of case studies at ING, a large-scale Dutch internationally operating bank. We collect and analyze real-world data from hundreds of agile teams and projects. We integrate expert knowledge with data-driven techniques to identify key factors and their interactions that influence on-time delivery in agile projects. By analyzing these factors, we gain valuable insights into the data and techniques needed to improve the predictability of software delivery. Using these insights, we develop models that effectively predict delays in user stories and epics, and optimize sprint plans tailored to team goals and delivery performance. Our findings can help practitioners identify and manage delay risks in agile settings, inform the design of automated tools for planning support, and contribute to a relational theory of software project management.

1.1 Background and Context

In this section, we provide background on agile software development as our research context. Additionally, we discuss related work in software effort estimation, software project risk management, and software analytics.

1.1.1 Principles of Agile Software Development

Traditional software development follows a sequential, phase-based approach where each stage must be completed before moving to the next [26]. This structured process emphasizes upfront planning and documentation, often leading to long development cycles with limited adaptability to changing requirements. In contrast, agile methods take an iterative and adaptive approach to software development [27]. Agile teams develop software incrementally in short iterations, enabling faster delivery, continuous feedback, and greater flexibility to accommodate changing priorities [28]. Each iteration involves designing, implementing, testing, and delivering a functional product increment, with feedback from one cycle informing the next.

The key distinctions between agile and traditional software development are illustrated in Figure 1.1 and are reflected in the four core values of the Agile Manifesto [29]:

- *"Individuals and Interactions Over Processes and Tools"*: Agile prioritizes communication and collaboration over rigid processes.
- *"Working Software Over Comprehensive Documentation"*: Agile values working software as the primary measure of progress. While documentation is important, the focus should be on delivering functionality that adds value to the customer.
- *"Customer Collaboration Over Contract Negotiation"*: Agile emphasizes ongoing customer involvement to ensure the software meets expectations.
- *"Responding to Change Over Following a Plan"*: Agile acknowledges that requirements change over time and encourages teams to adapt instead of adhering to a fixed plan.

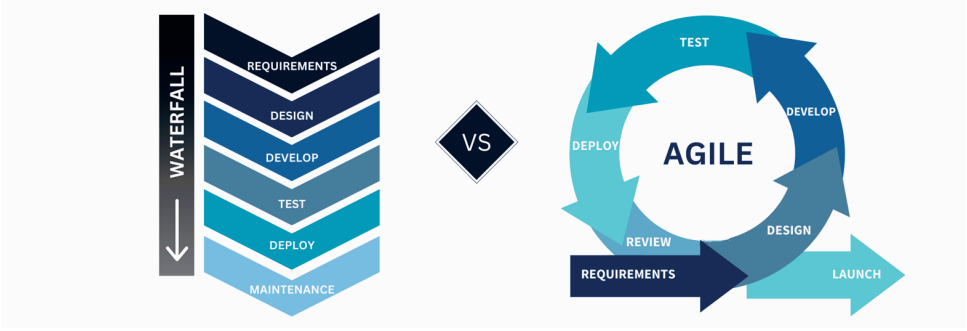


Figure 1.1: Traditional (waterfall-like) vs. agile software development

Since its introduction in 2001, the Agile Manifesto has driven widespread adoption of agile methodologies [30]. Frameworks such as Scrum [31], Kanban [32], and Extreme Programming (XP) [33] provide structured practices to implement agile principles effectively.

1.1.2 Agile Requirements Engineering

In agile projects, user requirements follow Leffingwell's five-level hierarchy [15], commonly adopted in the Scaled Agile Framework (SAFe) [34]. As shown in Figure 1.2, this hierarchy organizes work from high-level strategic goals to detailed development tasks. At the top are *strategic themes*, which define overarching business objectives. Within these themes, *epics* represent large functional goals spanning multiple iterations [13]. Epics are further broken down into *features*, significant functional components that deliver value to users. Features, in turn, consist of *user stories*, which capture specific requirements from an end-user perspective [14]. User stories are designed to be small enough to be completed within a single iteration. Finally, these stories are refined into *tasks*, which define concrete development activities and allow teams to track progress effectively.

1

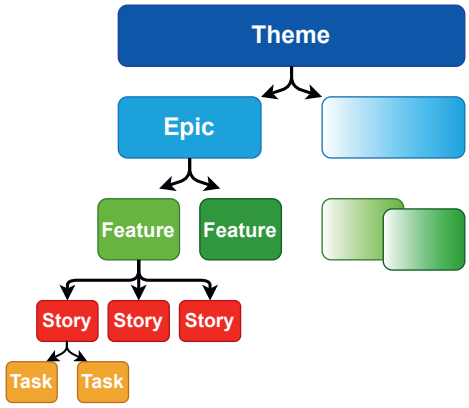


Figure 1.2: Leffingwell's five-level hierarchy for agile work breakdown

The Product Owner plays a key role in agile teams, acting as the link between stakeholders and the development team [13]. They are responsible for defining, refining, and prioritizing requirements to ensure alignment with business objectives. To manage and prioritize work, agile teams maintain a *product backlog*, a centralized repository of epics, features, user stories, and necessary fixes [35]. The Product Owner is responsible for backlog management, prioritizing items based on urgency, with the most critical placed at the top. The urgency of an item is determined based on immediate customer needs and project deadlines. As priorities shift, the backlog is continuously refined to keep the team aligned with evolving requirements. Agile teams use backlog management tools like Jira, Trello, and ServiceNow to manage backlog items efficiently.

1.1.3 Agile Planning Frameworks

Agile projects use structured planning frameworks to ensure alignment, adaptability, and continuous improvement [34, 36, 37]. These frameworks operate at multiple levels, typically relying on three key planning stages [13], as illustrated in Figure 1.3:

- **Release planning:** Covering a two- to six-month time frame, release planning focuses on epics and features [13], often involving multiple teams. It is a collaborative effort that establishes a road map, manages dependencies, and ensures that each release delivers tangible value. Product owners and project managers lead this process, while development teams provide input on dependencies and technical constraints.
- **Iteration planning:** Agile projects are divided into iterations, also referred to as sprints [38], which are short, time-boxed periods lasting one to four weeks. Before each sprint, the development team selects a subset of user stories from the product backlog to be delivered in that iteration. Through collaborative discussions, the team refines these stories into tasks for estimation [14].The team commits to completing a specific amount of work in the sprint. During the sprint, the team designs, implements, and tests the selected user stories to deliver a working product increment, such as a functional milestone.

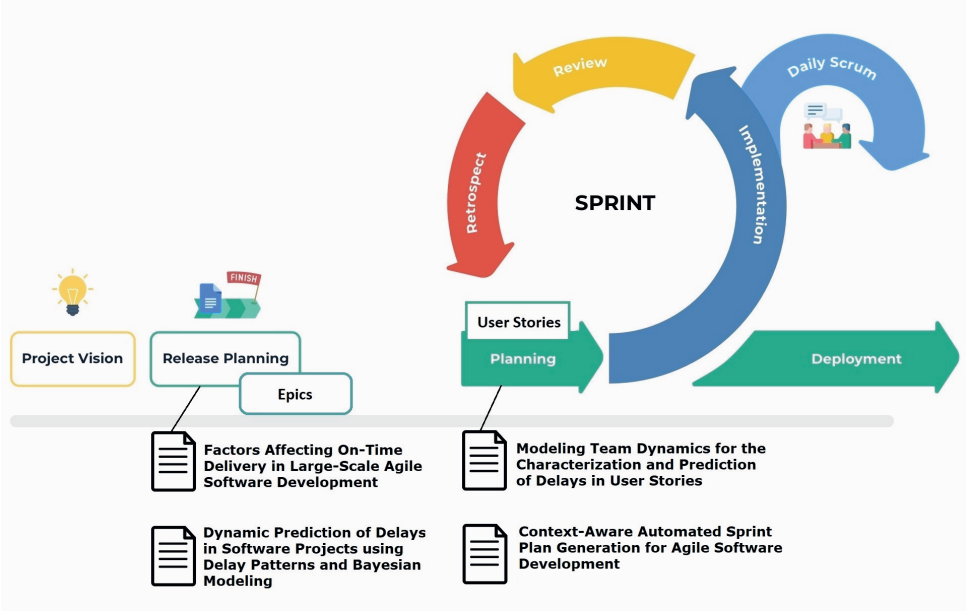


Figure 1.3: Stages of the agile development process. Agile planning occurs at three levels: release planning, iteration (sprint) planning, and daily planning. Annotations indicate corresponding articles of this thesis at the release and sprint planning levels.

- **Daily planning:** Daily stand-up meetings serve as structured touch points for daily planning. Team members review progress by discussing tasks completed yesterday, those planned for today, and any impediments they face. These meetings provide an opportunity for team members to synchronize their activities and plans.

These planning levels are interconnected and iterative, allowing teams to refine their approach based on feedback and evolving requirements throughout the project life cycle.

1.1.4 Software Effort Estimation

Effort estimation is a key activity in software project management, essential for planning and monitoring [39]. It involves predicting the effort — measured in time, resources, or cost — needed to complete a task or an entire project. Estimation accuracy impacts project success: underestimation can cause schedule and budget overruns, while overestimation may reduce an organization's competitiveness [40].

Research in software effort estimation spans several decades, highlighting its critical role in software engineering. In both the literature and this thesis, the terms effort estimation and effort prediction are used interchangeably. While effort estimation typically refers to expert judgment, effort prediction is commonly associated with data-driven or model-based forecasting. However, both terms refer to the same overarching concept of anticipating software development effort.

1

Factors affecting effort. Previous research has identified a large number of factors, referred to as effort drivers, that may influence software development effort. The accuracy of effort estimation methods depends on how well they select relevant factors and discard irrelevant or misleading ones [41, 42]. Trendowicz et al. [41] classified the most commonly used effort drivers into four categories:

1. Personnel factors, such as team capabilities and experience [6, 12, 43, 44].
2. Process factors, including the quality of methods and tools used [45–47].
3. Project factors, such as available resources and management activities [48, 49].
4. Product factors, including requirements analysis, design, and coding effort [50].

Expert-based and model-based methods. Effort estimation methods can generally be classified into expert-based and model-based approaches [6, 39]. Expert-based methods rely on human judgment to select relevant factors and remain the most widely used technique in both agile and traditional projects [6, 51]. However, these methods are inherently subjective, which can lead to inconsistencies and biases in estimation. Model-based methods, on the other hand, leverage historical project data to identify relevant factors from an initial dataset. While these methods offer a data-driven alternative, their effectiveness depends on the quantity and quality of available data, which can vary across projects. Since no single method is likely to be superior in all settings [6], recent studies suggest combining expert- and model-based approaches to mitigate their individual limitations [52, 53].

Machine learning models have gained popularity as an alternative to traditional model-based approaches. Common methods include regression analysis (e.g., [54, 55]), neural networks (e.g., [18, 25, 56]), fuzzy logic (e.g., [57–59]), Bayesian networks (e.g., [60, 61]), and evolutionary approaches (e.g., [62]). These models have shown promising results in predicting effort for software projects [25, 61, 63–65], as well as estimating the time required for bug-fixing and issue resolution [23, 66–70].

Human factors in effort estimation. Beyond the technical aspects of estimation, human and organizational factors play a significant role in shaping estimation outcomes. Prior work by Magazinius et al. [71, 72] highlights how both intentional and unintentional distortions influence software effort estimation. Estimates may be shaped by organizational pressures, stakeholder expectations, and team dynamics rather than purely technical considerations. For example, estimators may deliberately inflate or deflate estimates for strategic reasons, such as securing additional resources, managing stakeholder expectations, or avoiding unrealistic deadlines. Similarly, studies by Jørgensen and Grimstad [73, 74] have shown that software professionals may strategically adjust estimates to reduce projected costs and increase the likelihood of winning contracts. This behavior is particularly common in competitive bidding scenarios but can also occur whenever project selection is based on estimated effort. Such selection biases can lead to systematic underestimation, ultimately contributing to cost overruns and project delays. Understanding these human-driven distortions is essential for developing more robust estimation techniques that account for both technical and behavioral factors.

1.1.5 Effort Estimation in Agile Settings

Agile teams typically rely on expert judgment to estimate software development effort based on personnel- and project-related factors [10]. Estimation occurs at both the release and iteration planning levels [12, 13]. During release planning, teams provide high-level estimates for epics, which span multiple iterations and are initially estimated with low precision [75]. Teams use methods such as Relative Sizing or T-Shirt Sizing (S, M, L, XL) to estimate the relative effort of epics by comparing them to one another [76]. These estimates are then converted into sprints or months to establish an expected delivery date.

Iteration planning involves a more detailed estimation of the effort required to complete user stories. Teams break down epics into user stories and further decompose stories into development tasks [14]. Each story is assigned to a single developer, though multiple developers may work on its sub-tasks. Estimation of user stories occurs during sprint planning sessions, involving all team members in discussions [41]. Agile teams commonly use *story points* as a measure to estimate the relative effort, complexity, and risks of a user story [13]. One widely adopted method for estimating story points is Planning Poker [77]. In this process, each team member receives a set of numbered cards representing story points. After discussing the stories, members select a card reflecting their effort estimate. The team then reviews high and low estimates to reach a consensus on the final story point value.

To plan future sprints, teams measure their *velocity*, defined as the average number of story points completed in past sprints [13]. Historical velocity data helps teams forecast their capacity for upcoming sprints. Previous research has explored machine learning models for automating effort estimation, including predicting story points for user stories [18–20] and forecasting iteration velocity [24, 25].

1.1.6 Software Project Risk Management

Risk factors in software projects are uncertain events that pose serious threats to successful project completion [78]. Several studies have identified ineffective risk management as a major cause of project delays [79–83]. Risk management consists of two primary activities: risk assessment and risk control. Risk assessment focuses on identifying, analyzing, and evaluating potential risks and their implications for the project or organization [84]. Once risks are assessed, risk control implements measures to minimize or mitigate their impact.

Seminal work in the area of risk assessment has been carried out by Boehm's "Top 10 Software Risks" [84] and the Software Engineering Institute's "Taxonomy-Based Risk Identification" [85]. Current practices often rely on high-level guidance such as risk checklists and expert judgment. Common risk factors in software projects include unclear or changing requirements (e.g., [84, 86–88]), underestimation of project complexity (e.g., [89, 90]), unstable organizational environment (e.g., [91, 92]), lack of management support (e.g., [90, 93]), user commitment issues (e.g., [90, 94]), and personnel shortages (e.g., [84, 95]). Various studies have focused on measuring risk levels [89], categorizing risk types [90], and developing mitigation strategies [96]. Wallace et al. [97] classified software risks into six dimensions: organizational environment, user, requirements, project complexity, planning and control, and team risk. They analyzed the impact of these risk dimensions on overall project performance. Menezes et al. [8] identified 148 risk factors through a systematic literature review and classified them according to the Software Engineering Institute's Tax-

1

onomy [85]. Other studies (e.g., [98–100]) have used statistical analysis to evaluate risk factors in software projects. For example, Letier et al. [98] developed a statistical decision analysis model to assess uncertainty in software architecture decisions. Choetkiertikul et al. [101] applied machine learning to predict delay risks in issue reports.

1.2 Research Goal and Questions

This thesis aims to improve effort estimation and project planning in large-scale agile software development. To suggest meaningful improvements, it is essential to first develop an understanding of the factors and interactions that affect on-time delivery in agile settings. Prior research has identified a large number of factors that may influence software development effort [17], but their relative impact remains unclear. We lack an understanding of the relationships between these factors and how they influence on-time delivery.

We begin by examining high-level planning, where epics play a central role in managing interdependent software deliveries across teams and iterations. Delays at this level can disrupt broader project timelines, making it critical to identify the key factors and interactions that drive on-time delivery. Therefore, our first Thesis Research Question (TRQ-1) is:

> **TRQ-1:** What are the most relevant factors and interactions affecting the on-time delivery of epics?

Given the long-term nature of epics, it is essential to continuously reassess their overall delay risk to adapt to changes that occur during project execution. Existing delay prediction models in software engineering have a static character [25]; they are trained upfront and estimate the entire project based on predictor variables collected at the beginning of the project. They are unaware of changes occurring during project execution. While these models may be suitable for traditional, waterfall-like projects where predictor variables remain stable, they fail to capture the iterative and dynamic nature of agile development. In the field of transport, the concept of delay patterns has shown promising results in predicting evolving delays over time (e.g., [102, 103]). Building on this, our second Thesis Research Question (TRQ-2) focuses on developing a dynamic approach to predicting delays in epics:

> **TRQ-2:** How can the concept of delay patterns be adapted and applied to continuously predict overall delays in epic deliveries?

Next, we shift our focus to iteration planning, which centers on managing the delivery of user stories. At this stage, teams would benefit from actionable information about delay risks in user stories. This knowledge would enable them to identify problematic user stories and implement corrective measures (e.g., story splicing) to reduce the chance of delays. Due to the collaborative nature of agile projects, team-related factors such as team orientation, coordination, and work division can impact delivery performance [104–106]. Moreover, delivery performance can vary due to the influence of team changes, resulting

in team dynamics that may require incremental learning methods to be captured. This leads to our third Thesis Research Question (TRQ-3) on delay prediction in user stories:

> **TRQ-3:** Does the use of team features and incremental learning methods improve the accuracy of delay predictions in user stories?

Iteration planning, often referred to as sprint planning, is not only concerned with estimating user stories but also with prioritizing them effectively. While TRQ-3 focuses on predicting delay risks in user stories, another key challenge in sprint planning is deciding which stories should be prioritized and selected for development in the next sprint. This selection process is guided by various factors, known as prioritization criteria, which determine the order in which user stories are implemented. Although business value is typically considered the main prioritization criterion in agile methods, previous research suggests that teams apply prioritization criteria differently depending on their project context [107, 108]. However, how these criteria are weighted and applied in different settings remains largely unexplored [108]. Sprint planning is a complex and time-consuming process [109, 110], particularly in large projects where backlogs can grow to hundreds of user stories [111]. To support teams in this process, we aim to automate sprint planning by developing a model that estimates prioritization criteria and generates sprint plans tailored to the team's context. Our fourth Thesis Research Question (TRQ-4) is therefore two-fold:

> **TRQ-4: (a)** How does the importance of story prioritization criteria vary across project settings?, **(b)** How can teams' expertise and sprint history be integrated into a model to generate sprint plans that align with team goals and performance?

By addressing these research questions, this thesis aims to obtain valuable insights into the data and techniques required to advance the predictability of agile software delivery. **TRQ-1** lays the groundwork for understanding the factors and interactions that influence delays in epic deliveries. This knowledge can help software organizations proactively identify and manage delay risks in large-scale agile settings. Moreover, it contributes to the broader development of a relational theory of software project management. Building on this foundation, we apply these insights to design automated tools for delay prediction at both the epic level (**TRQ-2**) and the user story level (**TRQ-3**). TRQ-2 explores how delay risks evolve over an epic's life cycle, while TRQ-3 examines how team characteristics affect delays in user stories. Together, these insights can enhance the predictive power of existing effort estimation models. Finally, **TRQ-4** integrates team expertise and sprint history into a model for generating context-aware sprint plans. By improving the efficiency and effectiveness of sprint planning, this model provides organizations with a structured approach to balancing competing priorities and optimizing delivery outcomes. Ultimately, our research contributes to the field of agile software development by offering actionable insights, automating key planning processes, and refining effort estimation techniques. Our findings provide practical benefits to software organizations while also opening new avenues for future research on predictive analytics in agile project management.

1

1.3 Research Methodology

In this section, we provide an overview of the main research methods employed in our studies. The methodological framework of this thesis is rooted in *Empirical Software Engineering*, a sub-discipline dedicated to using empirical methods to investigate, evaluate, and improve software engineering practices [112]. This approach emphasizes data collection and systematic observations to generate evidence-based insights that inform both theory and practice in software development.

1.3.1 Case Study Design

To address our research questions, we conducted a series of *case studies* at ING (see Section 1.4). A case study is a research method used to investigate a phenomenon within its real-world context [113]. It involves an in-depth examination of a specific instance or case to obtain detailed insights that can inform theory development or practical interventions. Our case study design follows the framework established by Runeson and Höst [113], which defines a case study by its case, units of analysis, and research perspective. The research perspective refers to the study's stance on how knowledge is gained, such as positivist, interpretive, or critical. We adopt a single-case design, with ING serving as the focal organization. The units of analysis vary across studies (e.g., epics, user stories, or software teams) and are described in the respective chapters. From a methodological perspective, our case studies are conducted within a positivist research paradigm. Positivist case studies seek to establish objective findings through empirical observation, measurement, and hypothesis testing to generalize results to broader populations.

In our case studies, we employ qualitative and quantitative research methods, depending on their appropriateness to answer the research questions. Table 1.1 provides an overview of the research methods used in each study.

Qualitative methods. To gain insights into the perceptions of software experts, we apply qualitative research methods adapted from the social sciences, particularly Grounded Theory [114]. Grounded Theory is an inductive approach that systematically collects and analyzes data to develop theories or hypotheses derived directly from the data. Unlike methods that impose predefined categories, it allows patterns, themes, and concepts to emerge organically during the analysis, ensuring that findings are closely tied to the data itself. In our studies, we conduct surveys and interviews to collect qualitative data and apply open coding [115, 116] to systematically analyze it.

Table 1.1: Research methods used for each study and thesis research question (TRQ)

Study (addressing TRQ#)	Chapter	Survey	Interview	Quant. Analysis
Factors affecting on-time delivery (TRQ1)	2	✓		✓
Dynamic prediction of delays in epics (TRQ2)	3			✓
Delay prediction in user stories (TRQ3)	4			✓
Automated generation of sprint plans (TRQ4)	5	✓	✓	✓

Quantitative methods. For quantitative data collection, we employ data mining techniques to extract insights from historical data stored in software repositories. These repositories serve as an empirical source of ground-truth data, enhancing the precision and reliability of our findings [117]. In our studies, we extract data from Git (version control), ServiceNow (backlog management), and SonarQube (code quality analysis). While these are our primary data sources, other commonly used backlog management tools, such as Jira, Trello, and Azure Boards, offer similar functionalities. To improve the generalizability of our research, we gather data on a large scale from hundreds of teams and projects at ING. We apply data visualization techniques, statistical hypothesis testing, and probability-based approaches, including machine learning, deep learning, and Bayesian methods to uncover patterns and predictive insights from the quantitative data.

Mixed-methods design. In Chapters 2 and 5, we adopt a *mixed-methods approach* [118] in which we integrate qualitative and quantitative data to capture multiple perspectives on our research questions. This approach allows for triangulation, enhancing the validity of our findings by corroborating different data sources. In Chapter 2, we analyze factors affecting the on-time delivery of epics by comparing software practitioners' survey responses with quantitative data from backlog management and static code analysis tools.

1.3.2 Replicability and Open Science

The industrial data and source code used in our research cannot be publicly shared due to a Non-Disclosure Agreement. However, to increase external validity and encourage replication, we have made replication packages for our studies and models available on Zenodo. Table 1.2 provides an overview of these packages corresponding to relevant chapters. We have published our survey instruments and detailed descriptions of our model design and evaluation to support replication in different settings.

In line with TU Delft's Open Access policy, all articles that are part of this thesis are freely accessible via the `pure.tudelft.nl` repository. Links to these articles are provided in the respective bibliography entries.

Table 1.2: Overview of replication packages per chapter

Replication Package	Chapter	Zenodo DOI
Survey instrument, coding samples and data summary	2	10.5281/zenodo.11625946 [119]
Model summary and validation	3	10.5281/zenodo.11625842 [120]
Model comparison results	4	10.5281/zenodo.12206605 [121]
Survey instrument and demographics	5	10.5281/zenodo.11522834 [122]

1.4 The Case Company

In this section, we provide an overview of the case company studied and its approach to agile software development. ING, a large-scale, internationally operating Dutch bank, develops its software solutions in-house. With approximately 61,000 employees, including

1

17,000 software developers, and 53 million customers across 42 countries, ING is a major player in the financial industry [123]. Our research focuses on ING TECH, the bank's IT department, which consists of 295 development teams distributed across Europe, Asia, and North America. ING TECH is responsible for developing the bank's core banking applications and advisory services, used by millions of customers worldwide. The department covers a diverse range of products, varying in scale and application domain.

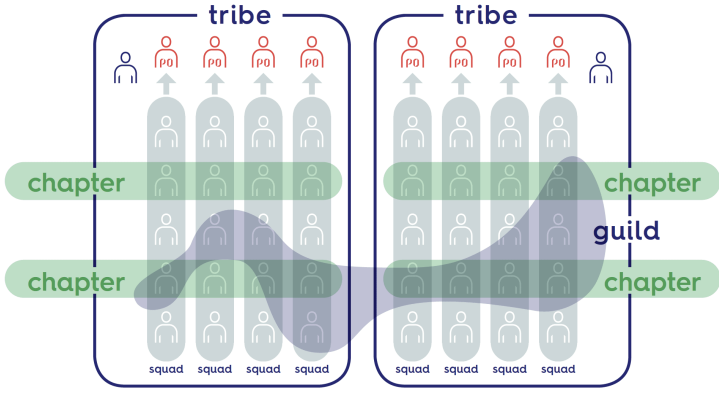


Figure 1.4: Spotify's 'Squads, Tribes and Chapters' model [124]

1.4.1 Agile Transformation

In 2011, ING initiated an agile transformation to shorten development cycles and improve responsiveness to changing customer needs. As part of this transition, the bank reinvented its organizational structure, moving from traditional functional departments to a fully agile model. To scale agile practices across its teams, ING adopted Spotify's Squads, Tribes, and Chapters model [124, 125], illustrated in Figure 1.4. Under this framework:

- **Squads** are autonomous, cross-functional teams of 5 to 9 members, each responsible for a specific client-focused mission. Squads operate with a high degree of self-organization, deciding what and how to build, as long as it aligns with their mission objectives. Each squad includes a Product Owner, who represents customer needs and ensures alignment across team activities.

- **Tribes** consist of multiple squads working toward interconnected objectives. A Tribe Lead coordinates activities and priorities across squads, while an Agile Coach supports squads by removing impediments and facilitating agile practices.

- **Chapters** group specialists with similar expertise (e.g., web development, quality assurance) across squads. Each Chapter Lead provides technical mentorship and guidance.

- **Guilds** span across multiple tribes and connect individuals with shared interests, fostering knowledge sharing and transparency within the organization.

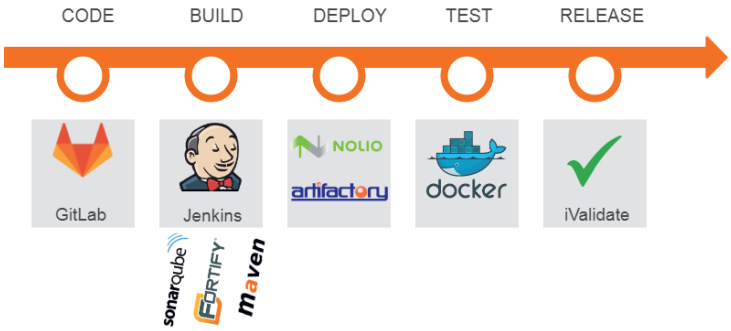


Figure 1.5: Continuous delivery pipeline at ING

1.4.2 Software Delivery Pipeline

Following this transformation, teams at ING shortened their development cycles from 2–3 months to 1–4 weeks. To enable rapid and reliable deployments, ING implemented a fully automated software delivery pipeline, used by all teams and illustrated in Figure 1.5. This pipeline integrates specialized tools across all stages of the software development life cycle. Based on the model proposed by Humble and Farley [126], the pipeline automates multiple tasks. When developers commit code, *Jenkins* (the continuous integration server) triggers automated processes for compilation, static analysis, and unit testing. As part of the pipeline, automated acceptance mechanisms verify build integrity by detecting compilation errors, failed test cases, and software quality issues. At ING, *SonarQube* [127] and *Fortify* [128] are used to assess static code quality and security vulnerabilities. Successfully built software artifacts are stored in Artifactory and deployed across different environments (testing, acceptance, and production).

1.4.3 Scrum at ING: Planning Sprints and Epics

All teams within ING TECH follow Scrum [38] as their agile framework. In Scrum, development occurs in sprints, which typically last 1–4 weeks. Each sprint begins with a sprint planning meeting, where teams select user stories from the product backlog, estimate their effort, and commit to completing them. Teams at ING use Planning Poker [77] and a fixed Fibonacci sequence of story point values for effort estimation. At ING, the guideline is that a one-point story corresponds to approximately 4 hours of work, while a two-point story requires 8 hours. This ensures additive and comparable estimates across teams.

During sprints, teams hold daily stand-ups to discuss progress, plan the day's activities, and identify any obstacles. At the end of a sprint, teams conduct a sprint review to demonstrate completed work to stakeholders and gather feedback. Scrum has two key roles: the Product Owner, responsible for defining and prioritizing the product backlog, and the Scrum Master, who ensures the team follows Scrum principles and facilitates continuous improvement.

At ING, epics typically span 1–4 business quarters (3–12 months). They are either delivered in full within a single quarterly cycle or incrementally through multiple software

releases. Tribe Leads oversee epic planning, collaborating closely with Product Owners to manage inter-team dependencies. Effort estimation for epics follows the T-Shirt Sizing technique [76], classifying epics as Small (S), Medium (M), Large (L), or Extra-Large (XL) to provide relative size estimates.

1.4.4 Case Selection and Generalizability

ING was selected as the case company because it serves as a representative example of large-scale agile software development. The bank's well-established agile practices and structured backlog management provide a rich setting to study the factors influencing on-time delivery and project planning. Additionally, ING has accumulated years of historical backlog data, offering a valuable foundation for predictive modeling.

From a generalizability perspective, ING is likely representative of other organizations in terms of team size, agile methodologies, and application domains. Its development teams work on a diverse range of software products, including banking applications, cloud services, and internal tools, making the findings relevant beyond the financial sector. Moreover, ING follows agile frameworks that are widely adopted in industry, including the Spotify model [124] and Scaled Agile Framework [125]. This further supports the broader applicability of our findings.

The high number of teams and extensive historical data may be more typical of large-scale organizations. However, we expect our findings on the identified delay factors and predictive modeling approaches to be transferable to other agile environments, regardless of company size. ING's strict security regulations as a financial organization may have influenced our findings on the relative importance of delay factors and prioritization criteria. We acknowledge this limitation and discuss its implications for generalizability in the Threats to Validity sections of the respective chapters.

1.5 Outline and Contributions

In this section, we outline the structure of this thesis by providing a summary of each chapter and its main contributions. This thesis is a compilation of independently published articles, with each chapter representing an individual article. To ensure coherence and maintain their integrity, we have made only minor modifications to these articles to create a unified thesis. We have preserved the fundamental structure of each article, ensuring that each chapter remains self-contained. All articles are accessible via TU Delft's `pure.tudelft.nl` repository, with links provided in their respective bibliography entries.

- Chapter 2 investigates the factors and interactions that affect schedule deviations in epic deliveries. We conduct a mixed-methods case study at ING to derive, confirm, and investigate influential factors. This study includes two rounds of surveys with 635 software practitioners to capture perceptions on key delay factors, their levels of impact, and the interactions among them. Additionally, we analyze software repository data from 185 teams to quantify and statistically model these factors, thereby corroborating the survey findings. We compose our findings in the form of a conceptual framework representing the factors and their interactions that affect the on-time delivery of epics (addressing **TRQ-1**). This chapter has been published as the TSE'21 article titled "*Factors*

Affecting On-Time Delivery in Large-Scale Agile Software Development" [129]. The main contributions of this study are:

  - A set of factors and their interactions affecting the on-time delivery of epics in large-scale agile development. We order the factors by their relevance.
  - A conceptual framework for on-time delivery that represents influential factors and their relationships. This framework suggests multiple paths for action that may improve the timeliness of software deliveries.

- In Chapter 3, we present a dynamic model for continuously predicting overall delays in epic deliveries. Unlike traditional static models, our approach leverages delay patterns and Bayesian inference to refine predictions over time. To identify delay patterns, we cluster time series of intermediate delay values recorded at designated milestones within epics. Our model detects these patterns on the go and incorporates them into its predictions, adapting to the context of the project phase and changes in team performance. We apply this approach to 4,040 epics from 270 teams at ING, identifying four distinct types of recurrent delay patterns. An empirical evaluation demonstrates that our model consistently outperforms static approaches and the state-of-the-art in software effort estimation (addressing **TRQ-2**). This chapter has been published as the ESEC/FSE'23 technical research paper titled "*Dynamic Prediction of Delays in Software Projects using Delay Patterns and Bayesian Modeling*" [130]. The main contributions of this study are:

  - A new approach for dynamically predicting overall delays using delay patterns and Bayesian inference.
  - An application of the approach at ING, resulting in the identification of four recurrent delay patterns.
  - An empirical evaluation of the approach and comparison to the state-of-the-art, demonstrating significant improvements in predictive accuracy.

- Chapter 4 presents a study on the effects of various aspects of teamwork on delays in user stories. We analyze historical backlog data from 571 teams and 765,200 user stories at ING to identify team-related factors characterizing delayed user stories. Based on these factors, we develop models that can effectively predict the likelihood and duration of delays in user stories. We evaluate our models using different feature sets and sliding window settings to investigate the potential of incremental learning methods. Our results show that the use of team-related factors and a sliding window approach leads to significant improvements in predictive accuracy (addressing **TRQ-3**). This work has been published as the ASE'21 technical research paper titled "*Modeling Team Dynamics for the Characterization and Prediction of Delays in User Stories*" [131]. The main contributions of this study are:

  - A set of team-related factors influencing delays in user stories. We rank the factors by their importance for delay prediction.
  - A new approach for predicting both the likelihood and duration of delays in user stories.

  - An empirical evaluation of the approach using different feature sets and learning methods, demonstrating significant improvements when incorporating team-related factors in a sliding window setting.

- In Chapter 5, we investigate how story prioritization criteria vary across project settings and present a model for automating sprint planning. We conduct a survey with 52 teams at ING to assess how they weigh prioritization criteria and how project characteristics influence these decisions. Findings reveal that urgency, sprint goal alignment, and business value are the most important criteria, with their influence varying based on project factors such as resource availability and client type. This highlights the need for contextual support in sprint planning. To address that need, we develop an optimization model that generates sprint plans tailored to team goals and performance. By integrating teams' planning objectives and sprint history, the model learns team-specific planning behaviors and adapts to unique team contexts. We evaluate our approach through both quantitative and qualitative analyses. Our quantitative evaluation, based on 4,841 historical sprints, demonstrates significant improvements in team alignment and sprint plan effectiveness. Our model outperforms the state-of-the-art and boosts team performance by generating plans that deliver 29% more business value, exhibit 14% stronger alignment with sprint goals, and reduce delay risk by 42% (addressing **TRQ-4**). Our qualitative evaluation, based on team interviews, confirms the usability and value of the model in practice. This chapter has been published as the ASE'24 technical research paper titled "*Context-Aware Automated Sprint Plan Generation for Agile Software Development*" [132] and received the ACM SIGSOFT Distinguished Paper Award. The main contributions of this study are:

  - A set of prioritization criteria ordered by their importance for sprint planning.
  - A context-aware optimization approach for generating sprint plans that align with team goals and performance.
  - An empirical evaluation of the approach and comparison to the state-of-the-art, demonstrating significant improvements in team alignment and sprint plan effectiveness.
  - A qualitative analysis with software teams, highlighting practical insights and identifying areas for future research.

  In Chapter 6, we revisit our research questions and discuss potential threats to the overall validity of this thesis. We conclude with practical recommendations for agile software organizations and outline promising directions for future research.

2

Factors Affecting On-Time Delivery in Large-Scale Agile Development

Late delivery of software projects and cost overruns have been common problems in the software industry for decades. Both problems are manifestations of deficiencies in effort estimation during project planning. With software projects being complex socio-technical systems, a large pool of factors can affect effort estimation and on-time delivery. To identify the most relevant factors and their interactions affecting schedule deviations in large-scale agile software development, we conduct a mixed-methods case study at ING: two rounds of surveys reveal a multitude of organizational, people, process, project and technical factors which we then quantify and statistically model using software repository data from 185 teams. We find that factors such as requirements refinement, task dependencies, organizational alignment and organizational politics are perceived to have the greatest impact on on-time delivery, whereas proxy measures such as project size, number of dependencies, historical delivery performance and team familiarity can help explain a large degree of schedule deviations. We also discover hierarchical interactions among factors: organizational factors are perceived to interact with people factors, which in turn impact technical factors. We compose our findings in the form of a conceptual framework representing influential factors and their relationships to on-time delivery. Our results can help practitioners identify and manage delay risks in agile settings, can inform the design of automated tools to predict schedule overruns and can contribute towards the development of a relational theory of software project management.

This chapter has been published as *E. Kula, E. Greuter, A. van Deursen, and G. Gousios. Factors Affecting On-Time Delivery in Large-Scale Agile Software Development, IEEE Transactions on Software Engineering [129].*

2

Late delivery and cost overruns have been common problems in the software industry for decades. On average, software projects run around 30% overtime [2]. This percentage does not seem to have decreased since the 1980s [3]. Even though effort estimation is at the heart of almost all industries, it is especially challenging in the software industry. This is mainly due to the fact that software development is a complex undertaking, affected by a variety of social and technical factors. The overall perceived success of a software project depends heavily on meeting the time and cost estimates [9]. Improving effort estimation is therefore a critical goal for software organizations: it can help companies reduce delays and improve customer satisfaction, while enabling them to efficiently allocate resources, reduce costs and optimize delivery [5, 6]. In spite of the availability of many estimation methods and guidelines [133, 134], on-time delivery in software development remains a major challenge. Prior research identified a large number of factors that may influence the software development effort [17], but which factors have the most impact is not clear. We lack an understanding of the relationships between these factors and how they impact on-time delivery.

Effort estimation is also a major challenge in agile software development. Prior work [12] has found that around half of the agile projects run into effort overruns of 25% or more. In agile settings, software is incrementally developed through short iterations to enable a fast response to changing markets and customer demands. Agile projects leverage short-term, iterative planning in which effort estimates are progressively refined [13]. A particular challenge involves combining the flexible, short-term agile planning setting with the business needs for long term planning of availability of large pieces of functionality (often referred to as "epics" [15]). Most agile teams heavily rely on experts' subjective assessment of team- and project-related factors to arrive at an estimate [10, 16]. However, these factors remain largely unexplored [16]; further analysis is required to investigate influential factors and how they impact delays in agile projects.

By identifying and investigating influential factors, we can obtain valuable insights on what data and techniques are needed to become more predictable at delivering software in agile settings. An identification of the most influential factors can help software organizations increase the effectiveness and efficiency of scheduling strategies by concentrating measurement and risk management activities directly on those factors that have the greatest impact on on-time delivery. Such knowledge can also guide future research on building and evaluating software effort estimation techniques, methods and tools. Furthermore, a deeper understanding of the interactions between influential factors can help in identifying the root causes of delays, and developing tools and guidelines that can assist software organizations in improving their on-time delivery performance.

The goal of this paper is to identify the most relevant factors and their interactions that affect schedule deviations in large-scale agile software development. To do so, we conduct a case study at ING. The teams at ING work with epics to manage interdependent software deliveries across multiple teams and iterations. We follow a mixed-methods approach in which we combine expert- with data-based strategies to derive, confirm and investigate factors that impact the timeliness of epics. We conduct a survey with 635 software experts from ING and analyze historic repository data from 185 teams and 2,208 epics to corroborate the survey findings. We extract proxy measures from repository data that we map to the perceived influential factors and analyze their importance in schedule deviations.

Throughout our study, the following two research questions guide our work:

- **RQ1. Factor identification:** Which factors are perceived to affect the timeliness of deliveries (RQ1.1), what is their perceived level of impact (RQ1.2), and what are the perceived types of interactions between these factors and on-time delivery (RQ1.3)?
- **RQ2. Factor validation:** How do the perceived influential factors impact schedule deviation in deliveries?

Our survey results show that requirements refinement, task dependencies, organizational alignment, organizational politics and the geographic distribution of teams are the factors that are perceived to have the greatest impact on timely delivery. We find that factors interact hierarchically: organizational factors interact with people factors, which in turn impact the technical factors. The technical factors are perceived to have a direct impact on the timeliness of software delivery. Our data analysis reveals that the project size, number of task dependencies, historical delivery performance, team familiarity and developer experience are the most important proxy measures that explain the schedule deviations in deliveries. By answering the research questions, we create a conceptual framework representing 25 factors and their interactions that are perceived to affect the timely delivery of software at ING. The main contributions of this paper are:

- A set of *factors and their interactions* affecting the timely delivery of software in large-scale agile development. We order the factors by their relevance.
- A *conceptual framework* of on-time delivery that represents influential factors and their interactions. This framework suggests multiple paths for action that may improve the timeliness of software deliveries.

2.1 Research Method

Our research method consists of an exploratory and confirmatory phase. In the exploratory phase, we developed and distributed a survey to software experts to identify factors that are perceived to affect the on-time delivery of epics (RQ1.1) and types of factor interactions (RQ1.3). In the confirmatory phase, we applied data triangulation to corroborate the respondents' perceptions and to extract more detailed insights into the effects of influential factors. We conducted a second survey with a different sample of software experts to order influential factors by their perceived level of impact (RQ1.2), and we performed regression analysis using repository data to validate the impact of factors (RQ2).

2.1.1 Collecting and Analyzing Survey Data

The main goal of the surveys was to gather the perceptions of software experts at ING on factors affecting the timeliness of epic deliveries and how much of an impact they have. To design and execute our surveys, we followed methodological guidelines from Kitchenham and Pfleeger [135], and Kasunic [136], for survey research in software engineering.

Survey Design

We developed two self-administered online surveys, which were composed of a mix of closed and open-ended questions. The first survey's purpose was to *identify influential*

factors and their interactions, and the second survey was used to *assess the perceived level of impact of each of the identified factors* from the first survey. The surveys were organized into two sections: a section aimed at gathering demographic information and a section targeting the research questions[1]. We kept the number of survey questions to a minimum as shorter questionnaires have been found to receive higher response rates [135].

The demographic sections of the surveys consisted of multiple-choice questions on the respondent's role, overall experience in software development and experience within ING. These demographic characteristics are important to assess the representativeness of participants [136] and they have been shown to influence the reasons given for effort estimation errors in related work [137]. The research related section of the first survey contained open-ended questions to gather unbounded and detailed responses on influential factors and types of factor interactions. For RQ1.1, we asked experts *which* factors affect the timeliness of their teams' epic deliveries. For RQ1.3, we included a follow-up open-ended question asking *how* the reported factors influence the timeliness of epic deliveries. In the second survey, we asked experts to rate the impact level of each identified factor from the first survey (RQ1.2). We used four-point Likert scale questions from "no impact" to "large impact" to get specific responses. We also provided a separate "not applicable" optional response in case a factor was not relevant to respondents. Here we also provided respondents with a write-in question to probe for additional factors in case any new ones might appear; we received 109 responses to that question. We reviewed the responses manually and found that they were rephrasing one of the 25 factors or identifying a subcase of one of the factors.

We have taken multiple measures to make the survey questions understandable by the respondents [135]. We included a brief paragraph on the survey's start page featuring the purpose of the survey and an overview of the types of questions presented in each section. Furthermore, we provided definitions of factors to participants in the second survey. The first two authors have considerable experience working in the company, and they were able to explain the factor definitions in vocabulary understood by the participants. Together with the last author, they also made sure that the survey questions were coherent and consistent [135]. To avoid leading questions and biasing respondents, we phrased and ordered the questions in a sequential order of activities [136]. The factors in the second survey were presented in random order to the participants to reduce ordering bias [138].

Survey Validation

After design, the survey instrument should be evaluated to display areas for improvement [135, 136]. We piloted both surveys with 25 randomly selected employees from ING TECH to refine the survey questions. The pilot versions included an additional open-ended question at the end of the survey asking respondents for feedback on the survey contents. The respondents' feedback allowed us to refine the survey questions. As part of the pilot run, we received 6 responses (24% response rate) for the first survey and 5 responses (20% response rate) for the second survey. No reminder emails were sent. The pilot of the first survey revealed that the wording of the survey question aimed at RQ1.3 was unclear. The question asked respondents about the types of relationships between factors, which respondents interpreted in different ways (e.g., sign of impact, causality versus correlation,

[1]The final survey instruments can be found in an online replication package [119]

direct or indirect link). Since we wanted to collect descriptive information about factor interactions and do the classification of relationships ourselves, we rephrased it as a more open-ended question in the final version of the survey. The initial version of the second survey disclosed that the names of some factors (e.g., task dependencies versus technical dependencies) were ambiguous to respondents. This prompted us to provide a list of factor definitions in the final version of the second survey.

Survey Execution and Sampling Strategy
Our target population was composed of the 2850 employees that belong to the 295 development teams at ING TECH. All these teams work with epic deliveries and are therefore relevant for our study. We received access to a mailing list containing all team members, which became our sampling frame. We were able to identify participants based on their email address and we had an overview of teams (team names) they belong to. This enabled us to determine members' participation and link survey responses to teams' repository data for triangulation in RQ2.

As recommended in survey guidelines [135, 136, 139], we performed simple random sampling to obtain representative samples from our population. For our final surveys, we excluded the 50 employees solicited in the earlier pilot surveys from our sampling frame. The final version of the first survey was distributed to 1400 employees (one half of the population) in October 2019. These employees were sampled uniformly at random across all teams at ING TECH. We received 298 responses (representing 237 teams), corresponding to a response rate of 21%. A majority (79%) of teams had one respondent, 16% had two respondents and remaining 5% had three respondents. The final version of the second survey was distributed to the other half of the population (another 1400 employees) in November 2019. This second group did not include employees solicited in the first survey. We obtained 337 responses (representing 241 teams), corresponding to a response rate of 24%. A majority (72%) of teams had one respondent, 18% had two respondents, 9% had three respondents and the remaining 1% had four respondents.

As per our sampling plan for the surveys, the participants were invited using a personal invitation mail featuring the purpose of the survey and how its results can enable us to gain new knowledge of delay factors in epic deliveries. Participants had a total of two weeks to participate in the surveys. To follow up on non-responders [135], we sent reminder emails to those who did not participate yet at the beginning of the second week.

Survey Data Analysis
The data we analyzed in this paper comes exclusively from the responses to the final surveys (i.e., the first survey deployed in October 2019 and the second survey deployed in November 2019). The "not applicable" responses were omitted from the analysis set. For the analysis of RQ1.2, we used descriptive statistics to order factors by their perceived level of impact. For the analysis of RQ1.1 and RQ1.3, we performed *inductive coding* to summarize the results of the open-ended questions. Coding samples are provided as examples in an online replication package [119].

Identifying influential factors. A common approach for transforming qualitative data into quantitative data is coding [115, 116]. For RQ1.1, we applied inductive coding (i.e., *inductive content analysis*) during two integration rounds to derive influential

factors from the open-ended survey responses. Each code in our coding scheme represents an influential factor. We coded by statement and codes continued to emerge till the end of the process. In the first round, the first and the last author used an online spreadsheet to code a 10% sample (30 mutually exclusive responses) each. They assigned at least one and up to four codes to each response. Next, the first and last author met in person to integrate the obtained codes, meaning that similar codes were combined or merged, and related ones were generalized or specialized if needed. When new codes emerged, they were integrated in the set of codes. The first author then applied the integrated set of codes to 90% of the answers and the last author did this for the remaining 10% of the responses. In the second round, the two authors had another integration meeting which resulted into the final set of codes. The final set contained three (13%) more codes than the set resulting from the first integration round. We computed percent agreement and Cohen's kappa [140] to assess inter-coder reliability on the final coding scheme. We measured substantial agreement between the coders: percent agreement = 86% and κ = 0.72. Then, together, the two authors grouped the factors into the five categories identified in the work of Chow et al. [9]. The resulting 25 codes and five categories are summarized in Table 2.1.

Classifying types of factor relationships. For RQ1.3, we analyzed open-ended survey responses to investigate the perceived types of relationships between influential factors and on-time delivery. We took the same approach as Jorgensen and Molokken-Ostvold [137]. We focused on direct, indirect and contributory relationships to make a distinction between simple, complex and condition-dependent types of interactions between factors. More discussions on types and interpretations of reasoning models can be found in the work of Pearl [141]. Possible interpretations of a factor X being a reason for schedule deviation are:

- There is a *direct* link between X and schedule deviation (*i.e.*, X is a direct reason for deviation). We classified a factor as having a *direct relationship* with on-time epic delivery if it is explained to be an immediate reason for schedule deviation. For example, "unmanaged dependencies" is a reason that may immediately lead to unplanned waiting time and thus delay in an epic.

- There is an *indirect* relationship between X and schedule deviation (*i.e.*, X leads to events that, in turn, lead to deviation). We classified a factor as having an *indirect relationship* with on-time delivery if it is explained to affect schedule deviation through other factors or events. For example, "lack of organizational trust" may lead to "errors during handoffs", which in turn may result in "unmanaged dependencies". "Lack of organizational trust" and "errors during handoffs" are both indirect reasons of different distance to the direct reason "unmanaged dependencies".

- The events leading to schedule deviation would have been harmless without X (*i.e.*, X is a *contributory reason* for schedule deviation). We classified a factor as having a *contributory relationship* with on-time delivery if it is described as a necessary condition for schedule deviation rather than a direct or indirect reason. Assuming that "unmanaged dependencies" is a direct reason for schedule deviation, a contributory

reason could be a "lack of a dependency management tool". That is, delays caused by "unmanaged dependencies" could have been prevented or reduced by effective dependency management.

For RQ1.3, we applied inductive coding during one integration round to derive a combination of (intervening) factors and their types of relationships to on-time epic delivery. We classified each reported relationship as a 'direct', 'indirect' or 'contributory' relationship using a separate code. Our interpretation of these relationships was based on the explanation of the respondent. For indirect and contributory relationships, we also coded the intervening factors that were mentioned. The first author performed the coding and classification for all answers. The last author did this for 20% of the answers. The resulting codes, including the intervening factors that followed from indirect and contributory relationships, matched with codes that were identified from open-ended responses to RQ1.1. No more new codes emerged in this process.

To evaluate inter-coder reliability, the first and last author met in person to compare the types of relationships identified. There were a few borderline cases in which a reported relationship would fit the indirect category as well as the contributory category. In such cases we tried to stay close to the formulation of the respondent. We classified a relationship as a contributory relationship only if an intervening factor was formulated as a necessary condition for the occurrence of another factor (e.g., using an if-then statement). If it was not phrased as a conditional statement, then we marked the relationship as an indirect one. Using Cohen's kappa [140], we measured substantial agreement between the coders: $\kappa = 0.69$ and percent agreement = 83%.

Survey Demographics

As mentioned earlier, the surveys contained a section aimed at gathering demographic information of the respondents, namely, their role within ING, total work experience at ING, total work experience in the software industry. A majority (66%) of the respondents self-identified as software engineer, while the rest identified themselves as manager or team lead (19%), product owner (7%), software architect (6%) or other (2%). The experience of the respondents at ING ranged from one year (24%) to more than 20 years (12%) with a median of between one and five years (41%). The experience of the respondents in the software industry ranged from one year (4%) to more than 20 years (24%) with a median of between 10 and 20 years (32%).

2.1.2 Collecting and Analyzing Repository Data

To quantitatively assess the impact of the perceived influential factors presented in Table 2.1, we extracted proxy measures from multiple data sources at ING that capture the respondents' intended meaning of the factors. In this section, we describe the datasets used and the linking process that we applied to the datasets. The primary goal of our regression model is to explain, rather than predict; we want to understand which proxy variables have a meaningful relationship with schedule deviations in epics. Therefore, we collected proxy variables that can be measured *before* and *after* an epic has been delivered. The mapping of proxy measures to perceived influential factors is shown in Table 2.3 and will be explained in Section 2.2.4.

Backlog Management Data

We extracted log data from *ServiceNow*, a backlog management tool used by a majority of teams at ING TECH.[2] The dataset consists of 3,771 epics delivered by 273 teams at ING TECH between January 01, 2017 and December 31, 2019. The dataset contains the following variables for epics: *identification number*, *creation date*, *planned start date*, *actual start date*, *planned delivery date*, *actual delivery date* and a textual *description*. We acknowledge that the planned delivery date of a delivery might change before actual development of the delivery is started. Therefore, we consider only the planned delivery date as scheduled on the day that the development phase is started. We provide an overview of all variables and their descriptions in an online replication package [119].

Code Quality Measurements

We extracted code quality metrics from *SonarQube*, a static code analysis tool in the software delivery pipeline at ING.[3] It is being used by most of the development teams at ING TECH. The tool offers a wide range of metrics related to code quality and unit tests execution. We extracted snapshots of SonarQube data of 190 teams that actively use the tool as part of their software delivery pipeline. Each snapshot is linked to a delivery in ServiceNow based on *team ID* and *time stamp* of when the measurement was done in SonarQube. If the time stamp of a SonarQube snapshot falls between the actual start date and actual end date of an epic in ServiceNow, the snapshot is considered to belong to the corresponding epic. Although SonarQube offers a wide range of metrics, we only consider the subset of metrics that are collected by all teams at ING TECH. These metrics allow us to measure and compare the code quality of epics in our analysis set. For each snapshot, we extracted the metrics that teams at ING TECH measure to assess their coding performance:

- *Coding standard violations*: the number of times the source code violates a coding rule.
- *Cyclomatic code complexity*: measured average cyclomatic complexity of all files contained in a snapshot.
- *Branch coverage:* the average coverage by tests of branches in all files contained in a snapshot.
- *Failed test ratio*: the number of failed tests divided by the total number of tests executed during the development phase of an epic.
- *Comment density*: the percentage of comment lines in the source code.

To account for differences in project size, we divided the metrics *Coding standard violations* and *Cyclomatic code complexity* by *Source lines of code*: the total number of lines of source code contained in a snapshot.

[2] https://www.servicenow.com/
[3] https://www.sonarqube.org/

Data Cleaning Process
To eliminate noise and missing values, we keep only the epics that meet the following conditions:

1. The *planned delivery date* and *actual delivery date* have been set.
2. The epic has been assigned to a team or group of teams.
3. The *description* field has been set.
4. The epic has been completed (i.e., if its *status* is set to *completed*).

We also removed outliers that exceed two standard deviations from the mean. For example, we removed six epics that had lasted longer than two years and that were, therefore, not representative for the rest of the dataset. The original dataset contained 3,771 epics. After linking and cleaning the data, the final dataset decreased to 2,208 epics from 185 teams for which all data are present. This group of teams overlaps with a majority (68%) of the teams that responded to the surveys.

Schedule Deviation Measures
There are a range of error measures used in effort estimation. Most of them are based on the Absolute Error (AE). Mean of Magnitude of Relative Error and Prediction at level l [142] have also been used in effort estimation. However, a number of studies [143–145] have found that those measures bias towards underestimation and are not stable when comparing effort estimation models. The *Balanced Relative Error* (BRE) [146] has been recommended as an alternative estimation accuracy measure. BRE is defined as:

$$\text{If Act - Est} \geq 0, \text{then BRE} = \frac{\text{Act - Est}}{\text{Planned duration}}$$

$$\text{If Act - Est} < 0, \text{then BRE} = \frac{\text{Act - Est}}{\text{Actual duration}}$$

where *Act* is the actual delivery date and *Est* is the planned delivery date of an epic (as reported on the start date of the development phase). $Act - Est$ calculates the difference in days between the actual delivery date and planned delivery date: a positive difference corresponds to underestimation (*Act* is later than *Est*), while a negative value corresponds to overestimation (*Act* is before *Est*). *Actual duration* is the time interval (in days) between the actual delivery date and start date of the development phase of an epic. *Planned duration* is the time interval (in days) between the planned delivery date and start date of the development phase of an epic. We assess the relative and absolute schedule deviation in epics using BRE and AE (measured in days), respectively.

Regression Analysis
A common approach for measuring the impact of a number of factors on estimation error is to use regression analysis. For RQ2, we used regression analysis to quantitatively assess the impact of combinations of perceived influential factors on the schedule deviation in epics. We extracted 35 proxy measures from backlog management data and code quality

measurements that can be mapped to the perceived influential factors. The proxy measures and their mapping to the influential factors are given in Table 2.3. An analysis of the proxy data revealed that it does not meet the assumptions for linear regression. Considering the need for an interpretable model in our explanatory study, we decided to go for a non-linear, spline-based regression modeling approach. We applied MARS [147]; a multivariate, piecewise regression technique that can be used to model complex relationships between a set of predictors and a dependent variable. We used the proxy measures as predictors and the measured BRE as dependent variable. MARS divides the space of predictors into multiple knots, i.e., the points where the behavior of the modeled function changes. This notion of knots makes MARS particularly suitable for problems with high input dimensions. The optimal MARS model is built using a backwards elimination feature selection routine that looks at reductions in the generalized cross-validation (GCV) criterion as each predictor is added to the model. This procedure makes it possible to rank variables in terms of their contribution to the GCV. We used the default MARS setting provided by the EARTH package in R.

2.2 Results

This section presents results on factors affecting delays in epic deliveries, derived from survey responses and repository data at ING. Example quotes from the survey are marked with a [rX] notation, in which X refers to the corresponding respondent's identification number. The perceived influential factors resulting from our manual coding process are underlined. The proxy measures that we map to the perceived influential factors are dashed underlined.

(RQ1) Factor Identification

2.2.1 (RQ1.1) Perceived Influential Factors

From the open-ended survey responses, we identified 25 factors that are perceived to affect the on-time delivery of epic deliveries. A list of these factors is shown in the left-hand column of Table 2.1. The factors are organized along the five dimensions identified in the work of Chow et al. [9]; organizational, process, project, people and technical.

Organizational Factors

This category of factors concerns the uncertainty surrounding the organizational environment in which an epic delivery takes place. Many respondents report the importance of organizational alignment for the on-time delivery of epics. A shared vision and mission are essential to ensure alignment between the implementation of an epic and its business strategy: "*A clear management vision creates focus and helps us align on business priorities and timelines across the company.*" [r216]

Another factor that is perceived to contribute to timely delivery is strong executive support. This includes the active involvement of management in strategy execution and the commitment of required resources. Respondent 285 explains that: "*It motivates us if management sufficiently participates in the preparation and performance review of delivery performance*".

In a related manner, respondents report delays related to organizational politics. Bureaucratic structures in the organization can hinder on-time delivery due to side steering:

"*Management should trust teams to come up with realistic timelines instead of pushing deadlines. This will prevent last-minute side steering and ad-hoc work.*" [r39] Other factors in this category that are perceived to hamper the on-time delivery of epics are the geographic distribution of teams and a lack of organizational stability (i.e., impact of organizational restructuring).

People Factors
People factors refer to qualities associated with a software development team that can affect the timeliness of deliveries. Factors that are perceived to contribute to the on-time delivery of epics are team stability (i.e., low team member turnover), strong skills and knowledge, team familiarity (i.e., the amount of experience individuals have working with one another) and team commitment to on-time delivery (i.e., motivation to deliver on-time). Teams that are more stable, skilled, familiar and committed to delivering epics on-time are perceived to deliver more often on-time. Moreover, respondents point to the importance of effective communication between teams, management and customers when it comes to technical problems and project delays.

Process Factors
This category of factors refers to the effectiveness and maturity of a software development team's way of working. The overall top mentioned factor is requirements refinement, which refers to the process of defining epics and dividing them into user stories. Missing or lacking details in the requirements is one of the main reasons for delay: "*Most of the time when we do not make the deadline, the team missed important information during refinement, which surfaced during the sprint.*" [r123] Here respondents also report the importance of frequent user involvement to manage user expectations and avoid delays caused by scope creep.

Another prominent factor featured in this category is regular delivery. Respondents explain the importance of having a short cadence for on-time delivery: teams that regularly deliver production ready software are perceived to be more predictable. Respondent 143 explains that "*Delivering software in shorter cycles enables our team to manage more complex projects and better predict our delivery capacity*". Some respondents indicate to feel more focused and effective at work when they limit the amount of work in progress at any given time.

Another important factor in this category is agile maturity, which stands for the ability of a team to become more agile over time. Respondents explain that they are able to improve their agility, and thereby, on-time delivery, over time through experience. Respondent 163 states that "*It helps to hold Scrum retrospectives and actually following up on their outcome. This allows teams to come up with ways to avoid, mitigate, or better handle impediments and other causes that impact delivery*".

Technical Factors
The technical category represents factors related to the quality of the source code artifact, and the effectiveness of technology and tools used to produce that artifact. Technical factors that are perceived to hamper timely delivery are poor code documentation, lack of code quality, bugs or incidents and insufficient testing. Well-defined coding standards are perceived to make teams more predictable in their deliveries: "*Higher quality standards*

will result in less incidents and less time spent on code refactoring in the future. This will, in turn, make our team more productive and predictable." [r334] Regarding testing, respondents mention to often fall behind schedule when preparing and executing time/resource-intensive types of tests, such as integration tests and performance tests. Such tests can lead to delays due to delayed availability of required infrastructure, unidentified dependencies and late identification of defects.

Another factor that is perceived to delay epic deliveries is a specific type of dependency — technical dependencies. Technical dependencies can occur among different software artifacts, e.g. source-code, architecture, hardware and tools. Teams at ING work with project-specific repositories and share codebases across teams within one application. As a result, teams at ING are hampered by source code dependencies across projects. Moreover, respondents perceive to be delayed by the unavailability or instability of technology and tools used for software development and software testing (unreliable infrastructure).

Project Factors

This category represents the inherent complexity and uncertainty of a software project. Task dependencies constitute a top mentioned factor that is perceived to delay epic deliveries. Task dependencies refer to dependencies among activities in the workflow of collaborating software development teams. Issues such as inconsistent schedules and unaligned priorities can cause delays for teams that collaborate in the same delivery chain: "*Task dependencies occur when teams are not end-to-end responsible for bringing software to production. They create the need for a significant amount of hand-offs.*" [r79]

Our respondents report several project characteristics that can affect on-time delivery: project size (i.e., the size of software, the size of development teams and project duration), degree of project newness (i.e., the innovative nature of a project) and project security (i.e., whether the project needs to go through resource-intensive security tests). Regarding the latter, respondents explain that business- and safety-critical applications are generally built and tested to much higher security standards, which may lead to unexpected delays in the quality assurance and security testing process.

2.2.2 (RQ1.2) Perceived Level of Impact

For each factor, respondents were asked to rate the level of impact using Likert-type choices of "no impact", "small impact", "moderate impact" and "large impact". The right-hand column in Table 2.1 shows the perceived level of impact of the factors as rated by the survey respondents. The *Top 2* percentage indicates the percentage of responses that rated the factor as having "large impact" or "moderate impact". The *Order* represents the order of factors by the weighted average of their impact level scores. Close to 60% of the respondents felt the factors are all moderately influencing their on-time delivery. Task dependencies, requirements refinement, organizational alignment, technical dependencies and regular delivery are the top 5 cited factors. They received more than 86% responses in the large and moderate impact category. Based on the weighted average of impact scores, requirements refinement (order #1), task dependencies (order #2), organizational alignment (order #3), organizational politics (order #4) and geographic distribution (order #5) are the top 5 rated factors. They received a weighted average impact score of 3.38 or higher. The impacts of the top 15 rated factors are perceived to have large or moderate

Table 2.1: Overview of 25 factors that are perceived to affect the on-time delivery of epics. The factors are organized along five dimensions: organizational, people, process, technical and project. *Percentage of respondents* is the percentage of survey respondents that mentioned the corresponding factor. The factors' *Level of impact* on delays was rated as 4 (large impact), 3 (moderate impact), 2 (small impact) and 1 (no impact). *Top 2* is the percentage of respondents that answered 4 or 3 for *Impact level*. *WA* is the weighted average of the Likert scale scores for *Impact level*. The factors' *Order* numbers are based on the ordering of the weighted averages.

Dimension	**Factor (RQ1.1)**	**Perc. resp.**	**Level of impact (RQ1.2)**			
			Distribution 1 2 3 4	Top 2	WA	Order
Organization	Organizational alignment	15%		90%	3.47	3
	Organizational politics	2%		86%	3.41	4
	Geographic distribution	2%		83%	3.38	5
	Executive support	2%		77%	3.18	14
	Organizational stability	4%		66%	2.91	20
Process	Requirements refinement	25%		91%	3.55	1
	Agile maturity	8%		84%	3.34	7
	Regular delivery	22%		87%	3.32	8
	Work in progress	6%		75%	3.05	16
	User involvement	9%		71%	2.95	19
Project	Task dependencies	16%		92%	3.49	2
	Project size	9%		84%	3.25	11
	Project newness	3%		83%	3.22	13
	Project security	4%		65%	2.83	22
People	Team stability	4%		85%	3.30	9
	Skills and knowledge	10%		83%	3.26	10
	Team familiarity	6%		76%	3.14	15
	Team commitment	4%		69%	2.97	18
	Communication	3%		47%	2.49	25
Technical	Technical dependencies	19%		89%	3.36	6
	Poor code documentation	3%		82%	3.23	12
	Unreliable infrastructure	7%		70%	3.03	17
	Bugs or incidents	6%		68%	2.89	21
	Lack of code quality	3%		65%	2.82	23
	Insufficient testing	5%		62%	2.73	24

impact by over 76% of the respondents. Communication (order #25) has lowest perceived impact but 47% of the respondents still rated it to have large or moderate impact.

Further analysis shows that respondents were quite consistent in their high ratings of most factors. The ratings for all factors have a standard deviation (SD) lower than 0.80, except for unreliable infrastructure (SD = 1.01), project security (SD = 0.99), lack of code quality (SD = 0.97), insufficient testing (SD = 0.96) and team commitment (SD = 0.96).

2.2.3 (RQ1.3) Perceived Types of Factor Relationships

We investigated open-ended survey responses to identify direct, indirect and contributory relationships between 25 perceived influential factors (from Table 2.1) and on-time epic delivery. As explained in Section 2.1.1, we focused on three types of relationships to distinguish between simple, complex and condition-dependent types of interactions between factors. An overview of the perceived types of relationships between factors and on-time epic delivery is shown in Table 2.2. Respondents mentioned three ways in which factors can have a direct impact on the timeliness of epics; factors can lead to unplanned **waiting time (WT)**, **necessary rework (NR)** or changes in **team effectiveness (TE)**. These are orthogonal types of effects. WT and NR are described to lead to delays, while increased (perceived) TE (i.e., a team's capacity to achieve its goals and objectives) is reported to help with on-time delivery. Table 2.2 shows the relevant type (NR, WT or TE) for each perceived direct relationship.

We found that the **organizational factors**, **people factors** and **technical factors** are perceived to interact **hierarchically**. The organizational factors have an impact on the people factors, which in turn affect the technical factors. From the organizational factors, executive support is perceived to have an indirect impact on timely delivery through team stability and team commitment. Respondents believe that strong executive support leads to more stable and highly motivated teams. Geographic distribution is associated with communication challenges and, thereby, reduced team effectiveness and delay. From the people factors, team stability is observed to be positively related to skills and knowledge and team commitment. Respondents explain that stable teams are more likely to develop competences over time and to take ownership of their work. Team commitment and communication are perceived to increase team effectiveness, and thereby, help with timely delivery. Skills and knowledge is reported to have an indirect impact through bugs or incidents and lack of code quality. Respondents point out that teams with more experienced members are faster at finding faults in software, and resolving unforeseen bugs and incidents. From the technical category, all factors are perceived to have a direct impact on on-time epic delivery. Respondents explain that problems related to technical dependencies, lack of code quality, bugs or incidents and insufficient testing can introduce necessary rework. Technical dependencies are also perceived to have an indirect impact on on-time delivery through *lack of code quality*. Respondents indicate that technical dependencies can introduce dependency problems, resulting in more complex and less maintainable source code.

In general, **process factors** are not perceived to relate to factors from other categories. One exception is the link between requirements refinement and task dependencies: respondents believe that an effective refinement process should reveal task dependencies, thereby enabling teams to minimize the likelihood of delays caused by dependencies.

Table 2.2: Types of perceived relationships between perceived influential factors and the timeliness of epic deliveries (RQ1.3): *Direct* relationship ↑, *Indirect* relationship →, *Contributory* link ⋆ or mentioned, but *no explicit relationship*. Factors can have three types of direct relationships with on-time delivery; they can lead to *necessary rework (NR)*, unplanned *waiting time (WT)* or deviations in *team effectiveness (TE)*. For indirect and contributory links, intervening factors are mentioned in the table. The percentages indicate the percentages of survey responses that mentioned the corresponding relationship.

Dimension	Factor	Direct ↑			Indirect →	Contributory ⋆	No explicit relationship
		NR	WT	TE			
Organization	Organizational alignment					Task dependencies, 12%	3%
	Organizational politics	2%					
	Geographic distribution				Communication, 2%		
	Executive support				Team commitment, 2% Team stability, 1%		
	Organizational stability	3%					1%
Process	Requirements refinement	25%			Task dependencies, 20%		
	Regular delivery				Work in progress, 20%		2%
	Agile maturity				Requirements refinement, 7%		1%
	Work in progress			5%			1%
	User involvement				Requirements refinement, 7%		2%
Project	Task dependencies		15%				1%
	Project size				Work in progress, 3% Technical dependencies, 2% Communication, 2% Task dependencies, 1% Insufficient testing, 1% Requirements refinement, 1%		1%
	Project newness			3%			

(Continued on next page)

(Continued from previous page)

Dimension	Factor	Direct ↑			Indirect →	Contributory ⋆	No explicit relationship
		NR	WT	TE			
	Project security					Insufficient testing, 3% Lack of code quality, 1%	1%
People	Team stability				Skills and knowledge, 3% Team commitment, 1%		1%
	Skills and knowledge				Lack of code quality, 6% Bugs or incidents, 4%		1%
	Team familiarity				Communication, 4%		2%
	Team commitment			3%			1%
	Communication			1%		Task dependencies, 2%	
Technical	Technical dependencies	13%			Lack of code quality, 6%		
	Poor code documentation			3%			
	Unreliable infrastructure		6%				1%
	Bugs or incidents	5%					1%
	Lack of code quality	3%					
	Insufficient testing	4%			Bugs or incidents, 1%		

Moreover, the most often reported relationship is the direct impact of requirements refinement on on-time epic delivery. Respondents report that refinement can prevent rework that is caused by unclear requirements. Regular delivery is perceived to lead to more consistency in the amount of work in progress, which in turn has a positive impact on team effectiveness and timely delivery.

The **project factors** are reported to have direct, indirect and contributory relationships with on-time epic delivery. Unresolved *task dependencies* are perceived to immediately result in delays through hand-offs and waiting times between dependent teams. *Organizational alignment* and *communication* are perceived to contribute negatively to the relationship between task dependencies and on-time epic delivery. Respondents explain that task dependencies can only stay unresolved and lead to delays in environments characterized by misaligned priorities and communication issues. Project size is perceived to be linked with factors across all dimensions. Larger epics are associated with more dependencies, parallel work, communication overhead, and increased testing and refinement effort. Innovative epics (project newness) are reported to involve time-intensive exploration activities and to be hampered by unforeseen obstacles, which can decrease team effectiveness and lead to delays. A higher level of project security is perceived to contribute positively to delays due to rework introduced by insufficient testing and lack of code quality.

Key findings from RQ1: We identified 25 factors that are perceived to affect the timeliness of epic deliveries. Requirements refinement, task dependencies, organizational alignment, organizational politics and the geographic distribution of teams are perceived to have the greatest impact on on-time epic delivery. The factors interact hierarchically; the organizational factors interact with people factors, which in turn impact the technical factors. The technical factors are perceived to have a direct impact on timely epic delivery.

2.2.4 Studied Proxy Measures

We formulate 35 proxy measures to map to the perceived influential factors. We extract the proxy measures from the datasets described in Section 2.1.2 to fit a regression model to quantitatively assess the impact of the perceived influential factors. Table 2.3 provides definitions for the proxy measures and their mapping to the perceived factors. Some proxies are self-descriptive; others are explained below. The proxies of the technical factors are described in Section 2.1.2. We take the median across teams or individual team members to produce measures that are representative of the group as a whole. For individual data, such as developer age, we take the median of team members working on an epic. For team data, such as team stability, we take the median of teams working on an epic.

We were not able to measure proxies for organizational alignment, organizational politics, executive support, communication quality, and technical dependencies. ING does not collect quantitative data on these factors. It was also not possible to collect such data ex post facto.

Table 2.3: Mapping of proxy measures to the perceived influential factors (from Table 2.1). The *Description* column provides a description of the proxy measure.

Perceived factor	**Proxy measure**	**Description**
Task dependencies	out-degree	Number of outgoing dependencies of an epic on other epics
Geographic distribution	global-distance	The maximum *Global Distance Metric* [148] (combining geographical, temporal, cultural distance) measured across teams working on an epic
Organizational stability	nr-changed-leads	Number of changed tribe leads during the current epic and previous epic
Team stability	stability-ratio	Median of the ratio of team members that did not change during the current epic and previous epic
Skills and knowledge	dev-age-team	Median of the number of years the developers working on an epic have been part of their team
	dev-age-ing	Median of the number of years the developers working on an epic have been working at ING
	dev-seniority	Number of senior developers working on an epic
Team familiarity	team-existence	Median of the number of years teams have existed in their current composition of team members
Team commitment	hist-performance	Median of the ratio of on-time delivered epics over all teams working on an epic
Requirements refinement	nr-updates	Number of times the epic's description field was updated before the start of the development phase
	state-ready	Whether refinement of an epic was completed before the start of its planning (binary)
Regular delivery	cycle-time	Sprint duration of teams working on an epic
	deviation	Median relative standard deviation from the regular cycle time during an epic
Agile maturity	team-point	Median of the total number of story points delivered by teams so far
	velocity	Median of the number of delivered story points per sprint during the current epic

(Continued on next page)

(Continued from previous page)

Perceived factor	Proxy measure	Description
Work in progress	parallel-epics	Median of the number of other epics that teams worked on simultaneously
	dev-workload-stories	Median of the number of stories assigned to a developer per sprint
	dev-workload-points	Median of the number of story points assigned to a developer per sprint
User involvement	acc-criteria	Whether acceptance criteria are specified for an epic (binary)
Poor code documentation	comment-density	The *comment density* measured at the start of an epic
Unreliable infrastructure	env-incidents	Number of environmental incidents that occurred during the development phase of an epic
Bugs or incidents	nr-incidents	Number of incidents that occurred during the development phase of an epic
	nr-unplanned-stories	Number of unplanned stories related to bug fixes or incidents that were added after the start of an epic's development phase
Lack of code quality	coding-violations	Number of *coding standard violations* measured at the start of an epic
	code-complexity	The *cyclomatic code complexity* measured at the start of an epic
Insufficient testing	branch-coverage	The *branch coverage* measured at the start of an epic
	failed-test-ratio	Number of tests that failed during the development phase of an epic
Project size	nr-stories	Number of planned stories assigned to an epic
	nr-sprints	Number of sprints assigned to an epic
	nr-points	Total number of planned story points assigned to an epic
	nr-teams	Number of teams working on an epic
	team-size	Median team size
	loc	Total *source lines of code* measured at the start of an epic
Project newness	novelty	Whether an epic contributes to the company's business transformation ambition (binary)
Project security	security-level	Whether an epic needs to pass a resource-intensive security testing process (binary)

(Continued on next page)

(Continued from previous page)

Perceived factor	Proxy measure	Description
Organizational politics		
Organizational alignment		
Executive support		No data available
Communication		
Technical dependencies		

Motivation for Mapping

Organizational factors: To quantify the global distance between teams, we calculate global-distance based on the Global Distance Metric proposed in related work [148]. We calculate the metric for pair-wise combinations of teams and take the maximum value. To assess organizational stability, we calculate nr-changed-leads; this has been shown to influence the success of software projects in earlier work [90].

Process factors: To assess the refinement quality of an epic, we calculate state-ready to determine whether the 'status' field of an epic was set to 'refinement ready' before the start of its planning phase. The intuition here is that epics that are not clearly defined will miss important information that is needed for planning and that could potentially result in delay. We also calculate nr-updates; the intuition here is that problematic epics will raise more comments and questions that could result in more updates and potentially a delay. To assess agile maturity, we calculate velocity and team-point; both have been shown to be representative of maturity in related work [13, 149]. We extract acc-criteria to measure user involvement in the definition of an epic. The backlog management data contains a special 'acceptance criteria' field that indicates whether teams consulted their customer(s) to define acceptance criteria.

Project factors: We quantify task dependencies as the outgoing degree (out-degree) of dependencies of an epic; this has been shown to predict delay in related work [21]. To assess project newness, we determine the novelty of an epic based on a special 'business goal' field in backlog management data. This field indicates whether an epic contributes to a 'business transformation', 'business continuity' or 'compliance-related obligation'. Epics that contribute to a business transformation are marked as novel, as they introduce significant change. To measure project security, we determine security-level, which indicates whether an epics requires a mandatory, resource-intensive security testing procedure before deployment.

People factors: To calculate dev-seniority, the number of senior developers working on an epic, we retrieve the expertise levels of developers as specified in backlog management data. At ING, developer expertise is evaluated using the five-stage Dreyfus model [150], which classifies individuals based on their experience in the software industry. According to this model, we identify developers with a Dreyfus skill level of 3 ('competent') or higher as senior. To assess a team's overall commitment to on-time delivery, we compute hist-performance, a metric capturing past performance trends. Prior research has demonstrated that team commitment is positively correlated with team performance and estimation accuracy in software projects [97, 151–153].

Table 2.4: The MARS model for the BRE values of epic deliveries (RQ2). Adjusted R^2: 0.672. The variables in the models are ordered by importance (see Figure 2.1).

Beta factor	Basis function	Coefficient	Relationship
BF0		0.1014	
BF1	Max(0, 8 - *nr-sprints*)	−0.0142	
BF2	Max(0, *nr-sprints* - 8)	0.0339	
BF3	Max(0, 4 - *out-degree*)	−0.0138	
BF4	Max(0, *out-degree* - 4)	0.0451	
BF5	Max(0, 0.61 - *hist-performance*)	0.1843	
BF6	Max(0, *hist-performance* - 0.61)	−0.2643	
BF7	Max(0, 2.66 - *dev-age-ing*)	0.0984	
BF8	Max(0, 0.93 - *team-existence*)	0.0242	
BF9	Max(0, *team-existence* - *0.93*)	−0.0099	
BF10	Max(0, 5 - *team-size*)	−0.0066	
BF11	Max(0, *team-size* - 5)	0.0248	
BF12	Max(0, *security-level* - 0.85)	0.1288	
BF13	Max(0, 8 - *nr-unplanned-stories*)	−0.0043	
BF14	Max(0, *nr-unplanned-stories* - 8)	0.0138	
BF15	Max(0, *changed-leads* - 2)	0.0344	
BF16	Max(0, 0.74 - *stability-ratio*)	0.3091	
BF17	Max(0, 19 - *nr-stories*)	−0.0038	
BF18	Max(0, *nr-stories* - 19)	0.0105	
BF19	Max(0, *nr-incidents* - 10)	0.0145	
BF20	Max(0, *dev-workload-points* - 12)	0.0129	

2.2.5 (RQ2) Factor Validation

How do perceived influential factors impact schedule deviation in epic deliveries?

To answer this research question, we applied MARS to quantitatively assess the impact of combinations of proxy measures on the BRE values in epic deliveries. Table 2.4 presents the optimal MARS model for BRE values based on all proxy measures presented in Table 2.3. The columns show, from left to right, the beta factor coefficients β_m denoted as BF_m, the basis functions selected as significant covariates in the model, the coefficient values estimated and a visualization of the relationship between the independent variable and BRE value. The value of the beta factor implies the magnitude of effect of the basis function (i.e., variable effect) on the BRE value. For the effect of each basis function, $max(0, x - t)$ is equal to $(x - t)$ when x is greater than t (the knot value); otherwise the basis function is equal to zero.

As shown in Table 2.4, the MARS model contains 20 basis functions and 13 proxy measures. The selected proxy measures represent all five of the factor dimensions. They are effective in explaining 67% of the variation in the BRE values of epic deliveries (adjusted R^2: 0.672.). Figure 2.1 provides a ranking of the proxy measures by order of importance. Proxies that have no impact on BRE are not shown. The importance is calculated as the relative importance of proxies in terms of reductions in the GCV estimate of the prediction error as each proxy measure is included. From this figure, we observe that nr-sprints, out-degree, hist-performance, dev-age-ing, team-existence and team-size have the greatest impact on schedule deviation in epics. Their importance values range from 15% to 21%.

Factors associated with delay. These factors show a rising relationship in the rightmost column of Table 2.4. From Table 2.4, beta factors BF1 and BF2 capture the nonlinear delay effect of nr-sprints, the most important variable in the MARS model. The number of sprints in an epic is positively related to BRE values with a knot at $t = 8$. The effect of nr-sprints can be explained as follows. The delays in epics tend to increase with a higher number of sprints. If an epic has fewer than 8 sprints, schedule deviation increases by 0.0142 per sprint (indicated by BF1). If sprints exceed 8, then deviation rises faster by 0.0339 per sprint (indicated by BF2). Other proxy measures that contribute to delay are out-degree, team-size, nr-unplanned-stories and nr-stories. They have a two-sided, positive relationship with BRE with a single knot. This indicates that delays increase with larger teams and with more outgoing dependencies, unplanned stories and planned stories. The proxies security-level, nr-changed-leads, nr-incidents and dev-workload-points can also contribute to delays: they show a right-sided, positive relationship with BRE with a single knot. This means that when these proxies exceed their corresponding knot value, epic delays increase. For example, the beta factor BF15 shows the nonlinear effect of nr-changed-leads on BRE which can be described as follows. If a team's tribe lead changed fewer than two times during the current and previous epics, it has a negligible effect on BRE values (indicated by BF0). However, if the number of changed leads exceeds two, BRE increases by 0.0344 per additional change. Similarly, if security-level is above 0.85, BRE increases by 0.1288. Since security-level is binary, this means epics requiring mandatory security testing have BRE values 0.1288 higher than those that do not.

Factors associated with on-time delivery. These factors show a downward relationship in the rightmost column of Table 2.4. As indicated by BF5-9 and BF16, the proxies hist-performance, dev-age-ing, team-existence and stability-ratio help with the timely delivery of epics. The delay in epics tends to decrease for teams that were less often involved with delay epics in the past and teams that have existed longer in their current composition. The delay can also decrease with a higher developer experience at ING and higher team stability up to the corresponding knot values.

Absolute deviation. Further analysis of the absolute deviations in epics showed that an overlapping set of 10 variables effectively explains 61% of the variation in the AE values. The proxy measure nr-teams emerged as statistically significant, while nr-unplanned-stories, nr-changed-leads, nr-incidents and dev-workload-points were not included in the model for AE values. Moreover, the proxies have slightly different importance values: out-degree (15), nr-sprints (14), team-existence (13), dev-age-ing (12), hist-performance (11) and nr-stories (10) have the greatest explanatory power for absolute deviation.

Key findings from RQ2: A set of 13 proxy measures is effective in explaining 67% of the variation in the BRE values of epics. The project size, number of task dependencies, historical delivery performance, team familiarity and developer experience at ING have the greatest explanatory power for schedule deviations in epics.

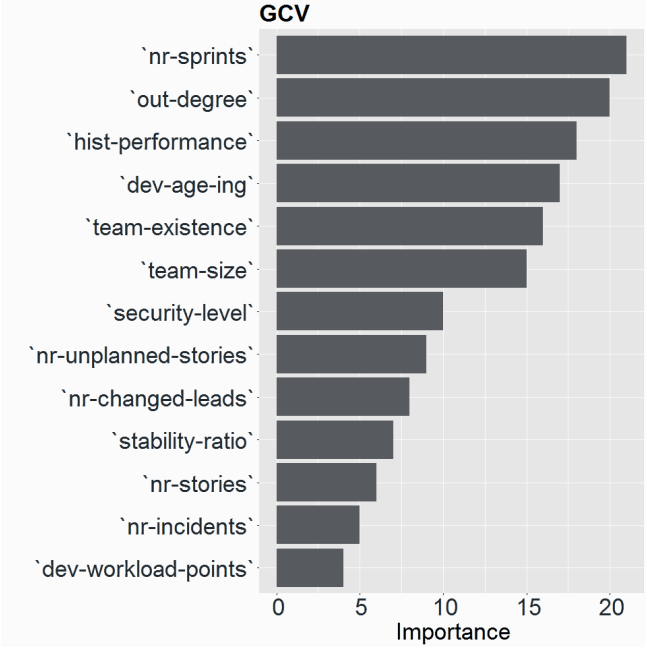


Figure 2.1: Importance of proxy measures in MARS model for BRE

2.3 A Conceptual Framework of On-Time Delivery

We organized the findings from our survey and regression analysis into a conceptual framework, presented in Figure 2.2. The framework captures the 25 perceived influential factors (from Table 2.1) and how they relate to on-time delivery. The connections between factors are derived from the reported types of relationships in Table 2.2. The directions are derived from the descriptions of factors in Section 2.2.1 and the relationships in Section 2.2.3.

Practitioners can use our conceptual framework to identify and manage the risks associated with on-time software delivery. For example, in our regression model, historical delivery performance is one of the most important proxy measures that affect schedule deviations in epics. Epics assigned to teams having a high percentage of delayed epics are at a high risk of being delayed. We recommend project managers to identify the teams that are involved with many delayed epics in the past, and consider allocating more time for their work or training them to successfully estimate effort and deliver on-time.

Our study also provides practitioners with a comprehensive list of factors and proxy measures that should be collected and analyzed to derive useful models that can be applied to improve on-time delivery. An assessment of the most significant factors would be a good starting point for further exploring influential factors in other settings. By weighing and analyzing the qualitative and quantitative attributes of factors, practitioners can choose the most important factors that influence their on-time delivery. Moreover, our analysis shows that expert- and data-based selection methods identified different (only partially overlapping) sets of relevant factors. Therefore, we recommend software practitioners to combine selection methods to extract more detailed insights and gain a better understanding of their software development processes.

The design of an effective strategy to improve on-time delivery must recognize the relationships between influential factors. The relationships in our conceptual framework are operationalized by reported relationships. We can therefore not reason about causal links between factors. However, our framework does enable us to form hypotheses that could lead to actionable insights and may suggest corrective actions to address the root causes of delay in similar development contexts. Our results suggest that addressing factors that have a direct relationship with on-time delivery should directly lead to an improvement in on-time delivery. For example, our survey respondents believe that a lack of code quality leads to necessary rework, and thereby, delay in epics. We therefore hypothesize that code quality improvements may reduce the likelihood of rework and improve the on-time delivery performance. Moreover, our results suggest that improvements in indirectly influential factors lead to improvements in intervening factors, which, in turn, improve the timeliness of deliveries. For example, executive support is perceived to have a positive impact on team stability and team commitment, which in turn leads to improved team skills, high-quality code and less bugs or incidents. We hypothesize that establishing stronger executive support may lead to more stable, committed and highly skilled teams that are better able to maintain code quality and resolve delays caused by bugs and incidents. The paths for action that can be inferred from our conceptual framework indicate that on-time software delivery requires attention across many factors, and that both social and technical factors may need to be addressed to enable continuous improvement.

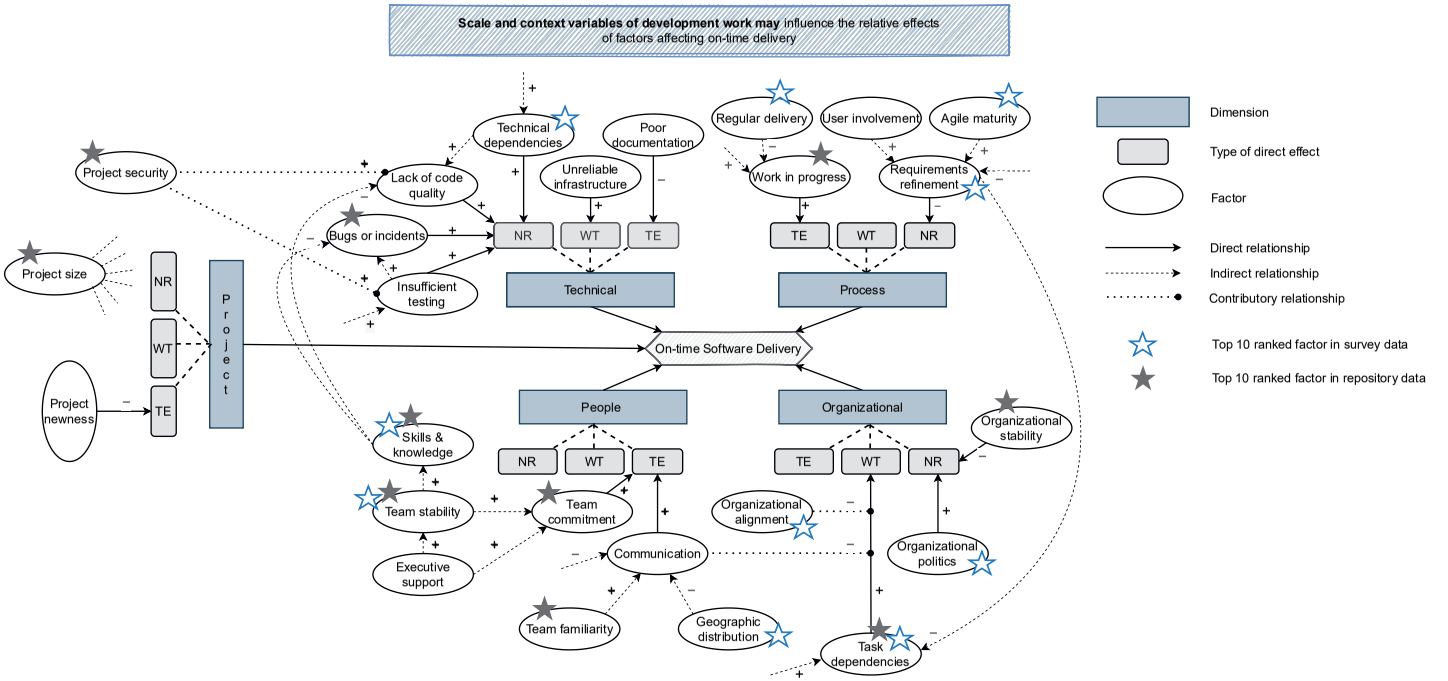


Figure 2.2: A conceptual framework of on-time delivery, based on survey responses and quantitative data from the case company. The dimensions/categories of factors are shown as blue rectangles. The 25 perceived influential factors are visualized as ellipses connected by directional arrows that indicate their perceived relationships. For readability purposes, the arrows starting from project size are partially omitted. For each relationship, the direction (*positive* + or *negative* –) is shown. Factors are found to impact delivery timeliness in three direct ways: they can lead to *necessary rework (NR)*, unplanned *waiting time (WT)* or changes in *team effectiveness (TE)*. Star symbols indicate the factors that are ranked among the top 10 most relevant factors in survey data or repository data.

2.4 Discussion

New influential factors. Our study has identified additional factors that influence on-time delivery and which, to the best of our knowledge, have not been covered in the current literature. We found that team familiarity is associated with timely software delivery. While prior research [154] has shown that familiarity is beneficial to team performance, it has not been investigated in the context of on-time delivery before. Our respondents believe that familiarity between team members improves the team coordination and helps in adapting to environmental changes. This indicates that for a better on-time delivery performance project managers should not only focus on keeping teams stable, they should also track and support teams to build familiarity over the long term. Moreover, agile maturity emerged as a new factor that is perceived to affect on-time delivery. The survey respondents rated this factor among the top 10 most influential factors. The survey responses point out that a growing agile maturity enables teams and ultimately the organization to continuously improve their on-time delivery performance.

Relevance of factors. The 25 influential factors we presented as part of RQ1 relate to previous research in effort estimation [41] and software project risk management [8]. Our results provide new insights into the relative effects of these factors. Our respondents perceive requirements refinement, task dependencies and organizational alignment to have the greatest impact on the timing of their deliveries. While requirements-related issues are top-cited risk factors in literature [8], task dependencies and organizational alignment have not received much attention. They have only been investigated in the context of scaling agile methods [155]. Further research is required to investigate the importance of these factors in the context of on-time delivery.

Prior work [10] has shown that team- and project-related factors are the most often mentioned effort drivers by agile practitioners. However, the relative importance of these factors has not been investigated before. The regression analysis we presented as part of RQ2 confirmed that the proxy measures of project factors (i.e., task dependencies, project size) and team factors (i.e., team commitment, team familiarity and skills & knowledge) have the greatest impact on schedule deviations. The project factors are found to have a slightly greater impact than task factors. We did not find a significant relationship between the code quality measurements and schedule deviations in epics. Poor code quality and documentation have also emerged as perceived influential factors in other studies [156, 157]. More in-depth studies are needed to corroborate the perceived impact of source code quality on on-time delivery.

Role of organizational environment. Our respondents reported the importance of several organizational factors for on-time delivery. The factors organizational alignment, organizational politics and geographic distribution were rated among the top 5 most influential factors in our survey. Among these, organizational politics and organizational alignment have been shown to impact project performance in related work [90, 97, 158] but they have not been identified as top influential factors before. Hence, our results suggest that different environmental aspects may play a larger role in on-time delivery than previously thought. Further research is required to investigate the impact of the organizational environment on on-time delivery in different settings.

Coordination challenges in large-scale agile. Our survey respondents indicated that several factors affecting on-time delivery are related to the challenges of adopting agile methods at the large scale of the case company. These factors include task dependencies, technical dependencies, geographic distribution and organizational alignment. Our respondents explained that their teams often depend on other teams and external parties for testing and deploying new software. This resonates with the findings by earlier work [155, 159, 160] that large-scale agile projects are more likely to be hampered by communication and coordination challenges. Further research is required to investigate the characteristics and impacts of different types of dependencies and task relationships in large-scale settings. Software organizations would benefit from mechanisms that make teams aware of inter-team dependencies, blockers and external dependencies that have the largest impact on their delivery time. Future research should study how existing coordination mechanisms are used in large-scale agile companies and how they can be improved to better support agile teams in delivering on time.

Incident management workflows. In line with earlier work [93, 161–164], we found that software deliveries are delayed by unexpected bugs and technical incidents. In our regression model, the number of unplanned stories and the number of incidents have a strong relationship with schedule deviation in epics. The disruptive nature of bugs and incidents calls for streamlined incident management processes and automated incident handling. Promising research in this direction has been carried out by Gupta et al. [165]. They used information integration techniques and machine learning to automatically link incoming incidents with configuration items. An interesting extension would be to leverage probabilistic modeling to predict the impact of an incoming incident on the time estimate of a delivery.

Multi-objective optimization for software delivery. We found that the security level of a software delivery is positively related to delay. Our respondents indicated that there is no tolerance for failure in some of the business-critical systems at ING. In highly regulated projects, engineers may need to decide to delay a delivery to increase time available for quality assurance and security testing. This alludes to a tension between delivery speed and the constraints imposed by regulations. New methods for rapid security verification and vulnerability identification could help organizations maintain agility. Related work [166–168] has focused on integrating security into agile methods and the challenges which this presents. Fitzgerald et al. [166] looked into the concept of *continuous compliance* and end-to-end traceability to support agile development processes in large-scale regulated environments.

The trade-off between timely delivery and security highlights a broader theme that predictable delivery is not the only factor that development teams in software organizations are trying to optimize. In reality, organizations deal with multiple objectives and look for optimal trade-off solutions that balance several criteria. Future research should investigate how the value of on-time delivery is measured and weighed in trade-offs in software industry.

A sociotechnical approach to improving on-time delivery. The perceived indirect and contributory relationships between influential factors (see Table 2.2) show that changing one factor may impact another, and that there are many non-technical factors that may also have an impact on on-time delivery. The results lead to some hypotheses that we discussed in Section 2.3, that may suggest corrective actions to improve the timeliness of software deliveries. The hierarchical interactions between factors indicate that the interplay between technical factors and on-time delivery is influenced by the social context of development work, as determined by organizational and people factors. This suggests that a healthy team culture and organizational environment can be used as intervening factors to resolve potential harmful effects of technical issues on on-time delivery.

Predicting delays in software deliveries. An interesting opportunity for future work is to incorporate the influential factors from our study into predictive models for delays in software deliveries. Currently, most software organizations rely on experts' subjective assessment to arrive at a time estimate for their deliveries. This may lead to inaccuracy and more importantly inconsistencies between estimates. Therefore, software organizations can benefit from predictive models that provide automated support for project managers in predicting the delivery time or delay of software deliveries. Existing models [21, 21, 67, 169] learn based on metadata (*e.g.*, type, priority) and/or textual features of the software task. Our results show that the predictive power of these models can be enhanced by incorporating the significant variables from our regression analysis. This will enable models to capture information about the software task as well as the environment in which the delivery takes place.

Our findings also suggest that an incremental learning approach might be beneficial for predicting delays in software deliveries. As unexpected events related to bugs, incidents, and a team members' departure can occur during the development phase (after a time estimate has been made), the prediction model should learn over time and adjust predictions based on newly acquired knowledge. A sliding window-setting could be used to model team dynamics and external changes. This boosts the ability of the model to learn and predict based on a team's recent delivery performance, and to forget older, irrelevant data. Initial work in this direction has been carried out by Abrahamsson et al. [25].

A relational theory of on-time delivery. A relational theory of on-time delivery would provide insightful and actionable information for project managers to design effective risk management strategies that improve on-time delivery performance. Our framework contributes in the advancement of a *relational theory* of on-time software delivery [170]. We identified and categorized influential factors (*descriptive theory*), and aim to specify the relations between factors to start to explain how they interact with each other. One potential research direction is therefore to investigate why factors are related, *e.g.*, through causality. This could be assessed by causal inference on time-series data, or through observations collected by experiments with development teams.

Guiding future research. We anticipate that our conceptual framework, and the set of factors we identified, can be useful for other researchers that study on-time software delivery. Our survey instrument and mixed-methods approach can be replicated to reveal

how specific factors impact on-time delivery in other settings. A consideration across organizations of different scale and context could lead to quite different factors that influence the results. Moreover, other social contextual settings, such as organizations with different levels of management support and organizational stability, could be considered to explore how the social setting affects the interplay between technical and social factors, and their impact on on-time epic delivery.

2.5 Threats to Validity

In this section, we discuss the threats to validity of our study and limitations with our conceptual framework. We used the checklist of Molléri et al. [171] to assess our surveys and identify threats to validity. The resulting scores from our assessment using the checklist can be found in an online replication package [119].

External validity: As with any single-case empirical study, external threats are concerned with our ability to generalize our results [172]. The company we studied employs thousands of software engineers and has significant variety in terms of the products developed, their size, and application domain (banking applications, cloud software, software tools). We performed random sampling and captured a range of roles and experiences, which may improve generalizability [171]. Even though our sample of survey participants is diverse, it is unlikely to be representative of software managers and engineers in general.

We conducted self-administered surveys, which may suffer from non-response bias [173, 174]. Our surveys were advertised as an "On-time Software Delivery Survey" and therefore could have led to over-representation of teams that deliver on-time. Developers from teams that are often delayed might have been less comfortable about participating in the surveys. Moreover, some of the factors presented in the second survey were geared at technical aspects of software development. This might have caused participants in non-technical roles to drop out, resulting in a bias toward software engineers. A limitation from our survey tool is that it does not record partially completed surveys. Hence, we do not know the dropout rate and cannot determine drop-out questions. This introduces a possible threat to external validity [171]. Our survey may have also been subject to self-selection bias, e.g., participants with strong opinions about delivery deadlines might have been more likely to participate in the survey. To mitigate non-response and self-selection bias, we sent personal invitations, kept the survey as short as possible and were transparent about the survey length. We also sent reminders to non-responders to increase the response rate and reduce the possibility of bias.

Although we control for variations using a large number of participants and projects, we cannot generalize our conclusions to other organizations. Replication of this work in different development settings is required to determine how work context influences (the perceptions of) factors affecting on-time delivery. Our findings indicate a trade-off between timely delivery and increased (security) testing and code refactoring effort. In a financial organization like ING there is no tolerance for failure in some of their business-critical systems. This may have influenced the factors we identified, making our findings likely to generalize more to software organizations with similar security regulations. Moreover, our results show several factors related to organizational fragmentation (e.g.,

dependencies, organizational alignment) which may be more common in large-scale organizations. In an effort to increase external validity and encourage replication, we have made our survey instrument available in an online appendix so that others can deploy it in different organizations and contexts [119].

Internal validity: We recognize that surveys can introduce biases and may contain ambiguous questions [171]. To mitigate these issues, we used terminology familiar to the target population and piloted the survey. We updated the survey instruments based on the validation results. In addition, we sought support from the second (confirmatory) survey and performed data triangulation. In the second survey (also piloted), no new factors emerged from the open-ended responses we solicited. In our survey design, we phrased and ordered the questions sequentially to avoid leading questions. We also randomized the order of factors to address order effects.

Our survey was not anonymous and therefore might have been subject to social desirability bias (i.e., a respondent's possible tendency to appear in a positive light) [175]. To mitigate this risk, we let participants know that the responses would be kept confidential and evaluated in aggregated form.

A factor that might have influenced our qualitative analysis is the bias induced by the involvement of the authors with the studied organization [171]. To counter the biases which might have been introduced by the first two authors, the last author (from Delft University of Technology) helped in designing survey questions and the manual coding of survey responses. In addition, we formally checked for reliability by computing intercoder reliability. Another risk of the coding process is the loss of accuracy of the original response due to an increased level of categorization. To mitigate this risk, we allowed multiple codes to be assigned to the same answer.

Construct validity: The goal of our first survey was to elicit influential factors and their interactions as experienced by engineers themselves. As the factors and types of relationships come from open-ended responses, and we rigorously assessed the expressed mechanisms through manual coding, we argue that they accurately represent the respondents' views. However, it should be noted that we did not verify whether participants can distinguish between affects. In the second survey we asked participants about the perceived impact of factors. To keep the survey short, each factor was measured by a single response item. Therefore, we could not test the reliability of participants' responses.

In our analysis of repository data, we consider data variables as constructs to meaningfully measure perceived influential factors. This introduces possible threats to construct validity due to measurement errors [176]. The proxy variables we measured may not capture the respondents' intended meaning of the concepts or constructs. Many factors, such as team commitment and organizational alignment, are quantifiable in principle but not directly measurable. For example, we measured the historical on-time delivery performance of a team as a reflective indicator of their commitment to on-time delivery. However, the commitment level of team members might not be reflected in their past delivery performance. A more common and direct way of measuring commitment is through psychological attachment instruments but it was not possible to collect such data ex post facto.

For the mapping of proxy measures to the perceived influential factors we had to find

acceptable trade-offs between the preciseness of proxy measures and the availability of data at ING. We acknowledge that for some perceived factors more precise alternatives can be found in related work. However, the repository data available did not cover equally precise data on all factors. Furthermore, it is possible that the data variables do not accurately represent reality. For example, we calculated time-related information on epics based on their planned and actual delivery dates in backlog management data. However, it might happen that teams close their deliveries too early or too late. We cannot account for the impact of poor record keeping on our results.

Transferability and credibility of our framework: We believe that the influential factors in the conceptual framework are likely transferable, to some extent, to other settings as they relate to previous research in effort estimation and project risk management. However, the specific results regarding the ordering of factors, factor relationships and regression analysis are bounded to the scale and context of ING. How factors impact on-time software delivery may vary according to scale and context factors of development work. It is noteworthy that we were not able to validate all factors as the repository data available did not cover all perceived influential factors. Our regression analysis does not exclude the importance of the non-included variables. Additional variables would probably have been included in our regression model if we had more data. We were not able to triangulate the relationships for RQ1.3. Replication of this work is required to validate the findings and reach more general conclusions. This might help enrich the framework.

2.6 Conclusions

Improving the timeliness of software deliveries is a challenge that is faced by many software organizations. In this paper, we identified and investigated the most relevant factors affecting delay in large-scale agile software development. We composed our findings in the form of a conceptual framework representing these factors and their interactions. The key findings of this study are:

1. Requirements refinement, task dependencies, organizational alignment, organizational politics and the geographic distribution of teams are perceived to have the greatest impact on timely software delivery.
2. Project size, number of dependencies, historic delivery performance, team familiarity and developer experience are the most important variables that explain schedule deviations in software deliveries.
3. Factors are found to interact hierarchically: organizational factors are perceived to interact with people factors, which in turn impact the technical factors. Technical factors are perceived to have a direct impact on timely software delivery.

Our conceptual framework suggests multiple paths for action that may improve the timeliness of software deliveries. Based on our findings, we identified challenging areas calling for further attention, related to the scalability of agile methods, inter-team dependencies, security concerns, the role of organizational culture, team stability and incident management. Progress in these areas is crucial in order to become more predictable at delivering software in agile settings.

3

Dynamic Prediction of Delays in Epics Using Delay Patterns and Bayesian Modeling

Modern agile software projects are subject to constant change, making it essential to re-asses overall delay risk throughout the project life cycle. Existing effort estimation models are static and not able to incorporate changes occurring during project execution. We propose a dynamic model for continuously predicting overall delay using delay patterns and Bayesian modeling. The model incorporates the context of the project phase and learns from changes in team performance over time. We apply the approach to real-world data from 4,040 epics and 270 teams at ING. An empirical evaluation of our approach and comparison to the state-of-the-art demonstrate significant improvements in predictive accuracy. The dynamic model consistently outperforms static approaches and the state-of-the-art, even during early project phases.

This chapter has been published as *E. Kula, E. Greuter, A. van Deursen, and G. Gousios. Dynamic Prediction of Delays in Software Projects Using Delay Patterns and Bayesian Modeling, ACM ESEC/FSE'23 [130].*

Schedule delays constitute a major problem in the software industry. Software projects run, on average, around 30-40% overtime [2, 177]. Ineffective risk management is one of the main reasons for delays in software projects [79, 82]. An important activity involved in risk management is delay prediction. Foreseeing delay risks enables project managers to take measures to assess and manage risks, make timely adjustments to the planning and reduce delay propagation. *Global* effort estimation models are the state-of-the-art in predicting overall delay for software projects [25]. Global models are trained upfront and estimate the entire project using predictor variables collected at the beginning of the project. These models have a *static character*: they capture the overall contribution of predictor variables to the total development effort and are unaware of changes occurring during project execution.

Global models are reasonable for traditional, waterfall-like settings where common predictors are known at the beginning of the project and do not change much throughout the project. However, this is not the case for modern, agile projects. In agile settings, projects (referred to as "epics") are incrementally developed through short iterations to respond fast to changing markets and customer demands [13]. Predictors proposed in previous work [17, 129], such as user requirements and task dependencies, can vary in value and relative impact during the execution of agile projects. Global models are not able to incorporate these changes due to their static character. An existing alternative is to use global models in an iterative manner (so-called *global iterative*) [25]. That is, applying the global model at different prediction times throughout a project using updated predictor values. This may lead to an improvement in predictive accuracy. However, the global iterative model is still not able to adapt to changes occurring during project execution. Agile projects call for the need of models with a *dynamic character*: models that are able to capture and adapt to changes in team performance and the impact of predictors during project execution.

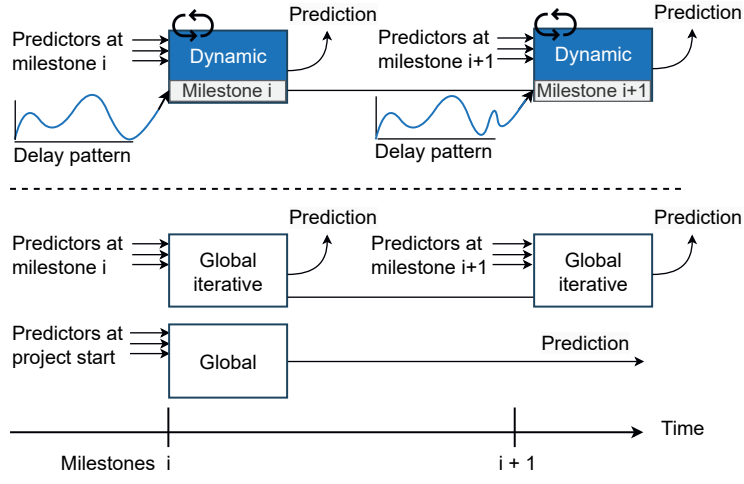


Figure 3.1: Global, global iterative and dynamic approaches to delay prediction over time

In the field of transport, prediction of overall delay is an important requirement for proactive control of traffic and the feasibility of timetable realisation [178]. Previous research in railway traffic (e.g., [102, 103]) and air transport (e.g., [179, 180]) has found that delays develop or propagate following certain patterns over time. A similar pattern in historic data can provide an estimation for the future development of delays. These studies detect patterns on the fly and use them for improving predictions of overall delay. It is not yet known whether this concept of delay patterns is applicable in the context of software development.

In this paper, we propose a *dynamic* effort estimation model for continuously predicting overall delay in agile projects. As visualized in Figure 3.1, the dynamic model extends global approaches by incorporating the context of the project phase (referred to as "project milestone") and modeling delay patterns when making predictions. The dynamic model is updated after each milestone using the predictor values collected for that milestone and the development of delay up until that milestone. The model captures the milestone-specific contributions of predictors to the total development effort and follows changes in team performance over time.

To develop our dynamic model, we use a Bayesian modeling approach. Bayesian models are able to learn from changes in the relative impact of predictors by updating their beliefs. We train the Bayesian model on time series of predictors and intermediate delays recorded across the milestones of a project's timeline. Similar to prior work in transport, we apply time series clustering to identify recurrent delay patterns. We apply our dynamic approach to real-world data from 4,040 epics and 270 teams at ING. We compare the performance of the dynamic Bayesian model with global approaches and the state-of-the-art baselines in software effort estimation.

An empirical evaluation of our approach demonstrates significant improvements in predictive performance, achieving on average 66–92% Standardized Accuracy and 0.19–0.04 Mean Absolute Error over time. The dynamic model consistently outperforms global approaches and the state-of-the-art, even during early milestones (i.e., 10–30% of project duration). The predictions of the dynamic model become substantially more certain and accurate over time. The main contributions of this paper are:

- A new approach to predict delay using delay patterns and Bayesian modeling (Section 4)
- An application of the approach at ING identifying four recurrent delay patterns (Section 5)
- An empirical evaluation of the approach and comparison to the state-of-the-art, clearly demonstrating a significant improvement in predictive accuracy (Section 6)

3.1 Related Work

Effort estimation models. Prior work has been done in building models for estimating effort of the entire project (e.g., [56, 62, 181]), a single iteration (e.g., [25, 61, 182]) and a single software task (e.g., a user story [18, 131] or issue report [21, 22, 183]). Existing models that estimate the total development effort are called *global* [25] and have a static

character. They make a single prediction using predictors collected at the start of the development phase. Global models can be applied in an iterative manner to obtain estimates at different prediction times throughout development. Choetkiertikul et al. [22] demonstrated this by applying their model for predicting delay risk at three different prediction times. They showed that the predictions become more accurate at later times since more information becomes available. Another study [183] identified patterns of abnormal behaviors causing project delays and used these patterns to predict the delay risk of issues. The patterns are derived as combinations of threshold-exceeding risk factors that can lead to schedule overruns.

Delay patterns in transport. Previous research in railway traffic (e.g., [102, 103]) and air transport (e.g., [179, 180]) has shown that delays develop or propagate following recurrent patterns over time. These patterns can provide information on the future development of delays. Artan and Sahin [102] used Markov chains to model patterns of delay deterioration, recovery and state keeping in train running times. Huang et al. [103] used a clustering technique to identify four types of delay patterns in train operations: decreasing delays, unchanged delays, small increasing delays and large increasing delays. They built a Bayesian Network model that uses the patterns in previous train stations to predict delay for upcoming stations. Oreschko et al. [179] detected specific delay patterns in flight arrival times with respect to the time of day and airport category. Jiang et al. [180] uses patterns of flight delay as input for a machine learning-based approach to delay prediction.

While delay patterns have been proven useful for delay prediction in transport, they remain unexplored in the context of software development. It is unclear whether and how delay patterns can be employed in software projects. Our study complements prior work by modeling delay patterns and using them as input for a dynamic approach to predict overall delay in software projects.

3.2 Bayesian Data Analysis

Recent works [184–186] identified the potential of Bayesian statistical techniques in software engineering research. Bayesian models are flexible, easy to interpret and provide a detailed probability distribution [184]. They are based on a uniform framework that applies Bayes' theorem to update prior beliefs about model parameters based on observed data. Bayesian models consist of three components [187]:

- *Likelihood*: A function that represents the probability of observing the data given a set of model parameters. It reflects the underlying data generation process. In the context of delay prediction, the likelihood captures the probability of observing a specific delay value or a set of delay values.
- *Priors*: Probability distributions that represent the initial beliefs or assumptions about the model parameters before observing the data. Priors allow incorporating existing knowledge about the effects of predictors.
- *Posterior*: The updated probability distribution that incorporates both the prior information and the likelihood of the observed data. It is obtained by repeatedly sampling values from the priors and applying Bayes' theorem using the likelihood. The posterior is used to make predictions about future observations.

3.3 Approach

Our overall research goal is to extract delay patterns and build a dynamic model that incorporates the patterns and the context of the project phase for continuously predicting the overall delay of an epic. This requires dividing an epic's timeline into designated milestones (Section 3.3.1) and tracking of intermediate delay and predictors across these milestones. The milestones should match the work pace of the organization and can be set accordingly at fixed time intervals or fractions of the planned project duration. It is a very common practice of agile teams to record the delivery status of their work items in a backlog management tool. Backlog data can be used to extract intermediate delay and predictors over milestones in the form of time series (Section 3.3.2).

To identify delay patterns, the time series of delay values over milestones need to be partitioned into groups of similar elements using clustering (Section 3.3.3). Hierarchical clustering or K-means can be used to identify and discriminate recurrent patterns. The dynamic prediction model is learned using the time series of the clustering output and predictor values (Section 3.3.4). For the Bayesian model, it is important to select the likelihood and tune the priors based on the dataset being used (Section 3.3.4). At each milestone, the updated variables are fed into the model to obtain a new, refined estimate and update the model's beliefs. This way the model learns and evolves with the epic over time.

3.3.1 A Unified Timeline of Project Milestones

To incorporate the context of the project phase, we present the timeline of an epic delivery as a sequence of regularly-spaced milestones. It is important to use a unified timeline so that delay values measured at the milestones can be aggregated across epics for pattern identification. Since teams working on an epic can follow different iteration lengths, we cannot use iterations or fixed time intervals. Instead, we define the milestones based on completion rate to evenly space them out along deliveries. The completion rate is based on the number of iterations completed compared to the total number of iterations planned. For example, an epic that consists of 20 iterations will achieve its 10% milestone after completing the initial two iterations. The total number of milestones used will determine the granularity of the collected time series and, therefore, the identified patterns. As progress updates are given at the end of every iteration, target milestones that cover the iteration length are used (usually 2-4 weeks [38]).

Milestones at ING. The average iteration length in our dataset at ING is 16 days. We performed our analysis with 10 milestones, which breaks most epics at ING down into intervals of two to three weeks with an average duration of 17 days. In total, 17% of the epics at ING consist of less than 10 iterations; we excluded those from our dataset to keep only the epics that have at least one iteration update available at every milestone (see Section 3.3.2). Each epic is divided into 10 milestones: every milestone is scored as 10% of the planned duration, so when a team reaches the third milestone of a task, their completion rate is equal to 30% and so on. When an epic's total number of iterations is not divisible by 10, we round the milestones off to the last completed iteration of their time frame. For example, when an epic consists of 18 iterations, its sixth milestone ($\frac{6}{10} \times 18 = 10.8$) will be measured at the end of the 10^{th} iteration. The milestones are connected by the

corresponding iterations and occur in a fixed sequence $j \longrightarrow k$, where $k = j+1$, $j = 1, 2, ..., 10$.

3.3.2 Data Collection

Backlog Data

To track changes in the intermediate delay and predictors over time, we need a backlog dataset that contains the history of epics. For each epic, this dataset has to include the *identification number*, *creation date*, *planned start date*, *actual start date*, *planned delivery date* and *actual delivery date*. At ING, we extracted log data from the backlog management tool *ServiceNow*. This dataset contained 7,463 epics delivered by 418 teams between January 01, 2017 and January 01, 2022.

Data Cleaning

To eliminate noise and missing values, epics with a status other than 'Completed' need to be removed. Epics that are not assigned to any team or have empty *Planned Delivery Date* and *Actual Delivery Date* fields also need to be filtered out. At ING, we chose to exclude the epics that consist of less than 10 sprints to keep only the epics that have at least one sprint update available at every milestone. In addition, we removed epics that exceed two standard deviations from the mean overall schedule delay of all epics. After linking and cleaning the data, the final ING dataset was reduced to 4,040 epics from 270 teams.

Delay Factors

The predictor variables can be obtained over milestones by extracting their values at the end of the last iteration that corresponds to a milestone. We extracted 13 predictor variables that represent factors affecting delays in epic deliveries. We identified these factors in previous work [129]. We used the same procedure to extract the predictor variables. Table 3.1 provides an overview of the predictors we collected and the influential factors they correspond to. For example, we model the delay factor *team familiarity* using the predictor variable *team-existence* that measures the amount of time team members have worked together in their current composition.

Measuring Schedule Deviation

We measure the overall schedule delay at the end of an epic using *Balanced Relative Error* (BRE) [146] as error measure. BRE has been recommended as an unbiased alternative to the commonly used Mean of Magnitude of Relative Error and Prediction at level l [143–145]. BRE is defined as:

$$\text{If Act - Est} \geq 0, \text{then BRE} = \frac{\text{Act - Est}}{\text{Planned duration}}$$

$$\text{If Act - Est} < 0, \text{then BRE} = \frac{\text{Act - Est}}{\text{Actual duration}}$$

where Act is the actual delivery date and Est is the planned delivery date of an epic. $Act - Est$ calculates the schedule deviation in days: a positive difference corresponds to a delay (Act is later than Est) and a negative value corresponds to on-time delivery (Act

is before *Est*). *Actual duration* is the difference (in days) between the actual delivery date and start date of an epic. *Planned duration* is the difference (in days) between the planned delivery date and start date of an epic.

To measure the intermediate delay of an epic at a given milestone, we select the last iteration corresponding to that milestone and calculate the total number of story points that are delayed to the next iteration/milestone. The total number of delayed story points represents the workload of user stories a team was unable to complete or resolve.

3.3.3 Clustering for Delay Pattern Discovery

To identify delay patterns, the time series of intermediate delay values recorded across milestones need to be clustered into groups of similar elements. In agile settings, intermediate delay can be measured based on the number of delayed user stories or story points. We measure the number of *Delayed Story Points* (DSP) as it is a more specific measure of the delayed workload. We calculate the delayed story points at a given milestone i as the number of story points that are delayed to the next milestone $i+1$. DSP represents the delayed workload at a particular milestone and is thus not cumulative. We normalize the DSP values per epic to make sure that the range of story point values cannot influence the clustering results. We divide the DSP values by the maximum number of story points that are delayed along the timeline of an epic.

To partition the time series data, we use hierarchical clustering with *Dynamic Time Warping* (DTW) as distance measure [188]. This approach has been shown to be appropriate for short time series [189]. DTW is a shape-based distance measure that finds optimal alignment between two time series that do not necessarily match in time or length. This makes DTW particularly suitable for epics that can differ in duration and sprint length, which is the case at ING. We use the Elbow method to select the optimal number of clusters k. This method calculates the total Within Cluster Sum of Squares (WSS) [190] for each k. The point of inflection on the WSS curve indicates the optimal number of clusters. Figure 3.2 presents the WSS curve for ING data and shows that a k value of 4 is optimal.

The application of our clustering approach to ING data resulted in four patterns that are discussed in Section 3.4. To characterize the clusters in terms of risk factors, we applied the Wilcoxon Signed Rank Test [191] for pairwise comparisons. This is a non-parametric test that makes no assumptions about underlying data distributions.

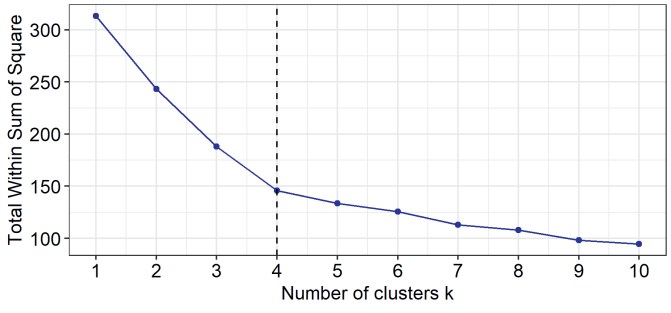


Figure 3.2: Elbow method and WSS curve for selecting the optimal number of clusters

Table 3.1: The 13 extracted predictor variables representing factors from [129] that affect delays in epic deliveries. The *Description* column provides a description of each variable.

Risk factor	**Predictor variable**	**Description**
Task dependencies	1. out-degree	Number of outgoing dependencies of an epic on other epics
Organizational stability	2. changed-leads	Number of changed tribe leads during the current and previous epic
Team stability	3. stability-ratio	Median of the ratio of team members that did not change during the current and previous epic
Skills and knowledge	4. dev-age-ing	Median of the number of years the developers working on the epic have been working at ING
Team familiarity	5. team-existence	Median of the number of years teams have existed in their current composition of team members
Team commitment	6. hist-performance	Median of the ratio of on-time delivered epics over all teams working on the epic
Work in progress	7. dev-workload	Median of the number of story points assigned to a developer per sprint
Bugs or incidents	8. nr-incidents	Number of incidents that occurred during the development phase of the epic
	9. unplanned-stories	Number of unplanned stories (related to bug fixes or incidents) that have been added to the epic
Project size	10. nr-stories	Number of planned stories assigned to the epic
	11. nr-sprints	Number of sprints assigned to the epic
	12. team-size	Median team size in the epic
Project security	13. security-level	The ratio of user stories in the epic that need to pass a security testing process

3.3.4 Bayesian Model Development

The main goal of our prediction model is to infer a probability distribution of BRE values across milestones. We use Bayesian statistical analysis to infer the probabilities and build the model in global and dynamic modes for comparison.

Different Modes of Model Development

We build and compare the Bayesian model using global, global iterative and dynamic modes of development. The differences between the models are visualized in Figure 3.1 and can be explained as follows:

- The **global** model solely uses the predictor variables as input and does not have a sense of time. It makes a single prediction of the overall delay based on predictors collected at the start of the project and does not update its BRE estimate throughout the project.
- The **global iterative** model is the global model used in an iterative manner (i.e., over milestones). We apply the global model at each milestone to obtain a new estimate of the overall delay based on the predictor values of that milestone. The model itself is not updated.
- The **dynamic** model is learned using the time series data of the clustering output and predictor values collected over milestones. This model incorporates the context of the milestone and thus has a sense of time. At each milestone i, the clustering model classifies the set of delay values across previous milestones 1 to $i-1$ into one of the four identified groups of patterns (producing a pattern label). To mimic a real prediction scenario, we set the values for future milestones $i+1$ to n to zero (unknown) in the input data for the clustering model. At each milestone, the pattern label and updated predictor variables are fed into the dynamic model to obtain a new estimate of the overall delay and update the model's posterior distribution.

Bayesian Modeling

We use Bayesian regression analysis to infer the probabilities that quantify delay risk and propagate uncertainty over time. We implemented our models in the statistical framework `Stan`[1]. We designed the models following the steps and guidelines for Bayesian data analysis in software engineering research [184–186]:

Step 1. Selecting a likelihood. The choice of a likelihood function depends on the type of data. The BRE values are proportional numbers between 0 and 1. In total, 42% of the BRE values in the ING dataset are zero (corresponding to on-time delivered epics). The data does not contain BRE values of one; the maximum BRE in our dataset is 0.83. A common choice for modeling proportional data is the Beta distribution likelihood [192]. Beta models are highly flexible and can take on all sorts of different shapes. To account for the zero values in the ING dataset, we selected the Zero-Inflated Beta distribution [193], relating predictors to outcome, as shown in Eq. 3.1. The Zero-Inflated Beta distribution depends on a mean μ and precision ϕ, like in a regular Beta, but it may produce a BRE of

[1]https://mc-stan.org/

zero with probability α in each draw from the distribution. We used a logit function for μ and α to translate them back to the log-odds scale of the (0,1) scale. We assume that all predictor variables may affect the parameters of the model (Eq. 3.2–3.4).

Step 2. Setting priors. To apply Bayes' theorem, we need to define priors for the model's parameters. A common approach, which works well in most cases, is a weakly informative prior [194], such as a normal distribution with zero mean and moderate standard deviation, as shown in Eq. 3.5 and 3.7. Such a prior does not bias the effect that the predictors may have towards positive or negative values, and it still allows for a wide range of parameter values. We set a Cauchy distribution (Eq. 3.6) for the β_ϕ parameters, which is a common choice for dispersion parameters [195]. To check what the combination of priors implies on our outcome, we sample from the priors only. This is called *prior predictive checks* (see Figure 3.3a).

The overall definition of the dynamic model is given in Eq. 3.1–3.7.

$$\text{BRE}_i \sim \text{Zero-Inflated Beta}(\mu_i, \phi_i, \alpha_i) \tag{3.1}$$

$$\begin{aligned}\text{logit}(\mu_i) = {} & \beta_{\mu_1} \cdot \text{out-degree} + \ldots + \beta_{\mu_{13}} \cdot \text{security-level} \\ & + \beta_{\mu_{14}} \cdot \text{milestone} + \beta_{\mu_{15}} \cdot \text{DSP} + \beta_{\mu_{16}} \cdot \text{pattern}\end{aligned} \tag{3.2}$$

$$\begin{aligned}\log(\phi_i) = {} & \beta_{\phi_1} \cdot \text{out-degree} + \ldots + \beta_{\phi_{13}} \cdot \text{security-level} \\ & + \beta_{\phi_{14}} \cdot \text{milestone} + \beta_{\phi_{15}} \cdot \text{DSP} + \beta_{\phi_{16}} \cdot \text{pattern}\end{aligned} \tag{3.3}$$

$$\begin{aligned}\text{logit}(\alpha_i) = {} & \beta_{\alpha_1} \cdot \text{outdegree} + \ldots + \beta_{\alpha_{13}} \cdot \text{security-level} \\ & + \beta_{\alpha_{14}} \cdot \text{milestone} + \beta_{\alpha_{15}} \cdot \text{DSP} + \beta_{\alpha_{16}} \cdot \text{pattern}\end{aligned} \tag{3.4}$$

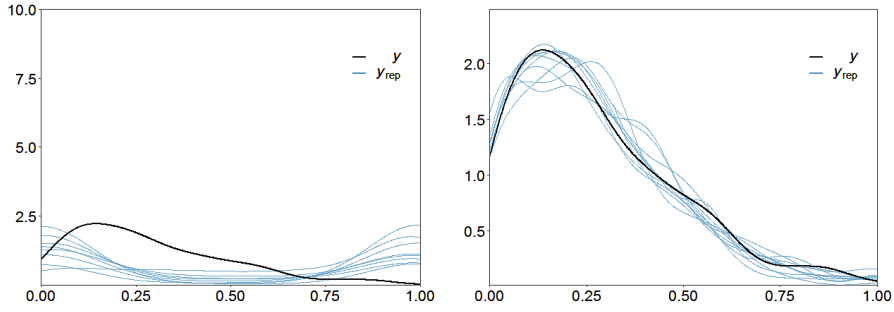
$$\beta_{\mu_1}, \ldots, \beta_{\mu_{16}} \sim \text{Normal}(0, 1) \tag{3.5}$$

$$\beta_{\phi_1}, \ldots, \beta_{\phi_{16}} \sim \text{Cauchy}(0, 1) \tag{3.6}$$

$$\beta_{\alpha_1}, \ldots, \beta_{\alpha_{16}} \sim \text{Normal}(0, 1) \tag{3.7}$$

The 'pattern' predictor in Eq. 3.2–3.4 stands for the delay pattern label as classified by the clustering model. The global and global iterative models follow the same design, except that they do not include the milestone and pattern label as predictors.

Step 3. Sampling. For sampling, we used the Hamiltonian Monte Carlo implementation provided by Stan. To improve the efficiency of sampling, we centered and scaled all predictor variables. Once the model has been sampled, we check diagnostics to ensure that we have reached a stationary posterior distribution. No warnings on divergent transitions and low E-BFMI values were reported [196]. Moreover, the $\hat{R}$ diagnostic was consistently less than 1.01 and the ESS value was higher than 0.2. This indicates that the Markov chains mixed well [197]. To check if the model fits the data, we sample from the priors with data. This is called *posterior predictive checks* (see Figure 3.3b). A summary of the model can be found in an online replication package [120]. At the 95% level, all predictors have a significant effect.



(a) Prior predictive check

(b) Posterior predictive check

Figure 3.3: Density overlays of predictive prior and posterior draws (visualized as light blue lines) versus the real data (shown as the dark blue line). The combination of our priors (left plot) shows that we assign more probability mass to low and high BRE values. After making use of the data (right plot) we see a good model fit: the light blue lines are covering the dark blue line.

Step 4. Model checking. To check for overfitting, we test whether any model making simpler assumptions about the data performs comparably or better than our model with the Zero-Inflated Beta distribution (M_{ZIB}). We compare M_{ZIB} with simpler models in terms of expected log predictive density (ELPD) using leave-future-out cross-validation [198, 199]. The models are conditioned on two years of historical data (covering the epics from 2017 to 2019) using the recommended threshold of 0.7 for the Pareto k estimates [199]. The results of our analysis can be found in an online replication package [120]. The results show that M_{ZIB} performs significantly better than other, simpler models and thus fits the data better while avoiding overfitting.

3.4 Delay Patterns at ING

Using the Elbow method, we determined $k = 4$ as the optimal number of clusters (see Figure 3.2). Therefore, we obtained four clusters representing delay patterns in epics at ING. Figure 3.4 visualizes the centroids and the 25th and 75th percentiles of the cluster delay distributions. The epics are grouped together with low mean variance (Var) around the cluster centroids (Var $C1 = 0.07$, Var $C2 = 0.11$, Var $C3 = 0.08$, Var $C4 = 0.12$), highlighting recurrent patterns.

Characteristics of clusters. Table 3.2 summarizes the statistics of the predictor variables for each cluster's epics. The confidence intervals are included in an online replication package [120]. We used the Wilcoxon test for pairwise comparisons (Bonferroni corrected) to identify the factors for which clusters are significantly different from the other three clusters (highlighted with an $*$ in Table 3.2). These factors characterize the epics exhibiting one of the four recurrent patterns. Even though we cannot reason about causal links between the factors and patterns, the results of the analysis enable us to form hypotheses on the causes of delays. Testing such hypotheses could lead to actionable insights and suggest delay mitigation measures.

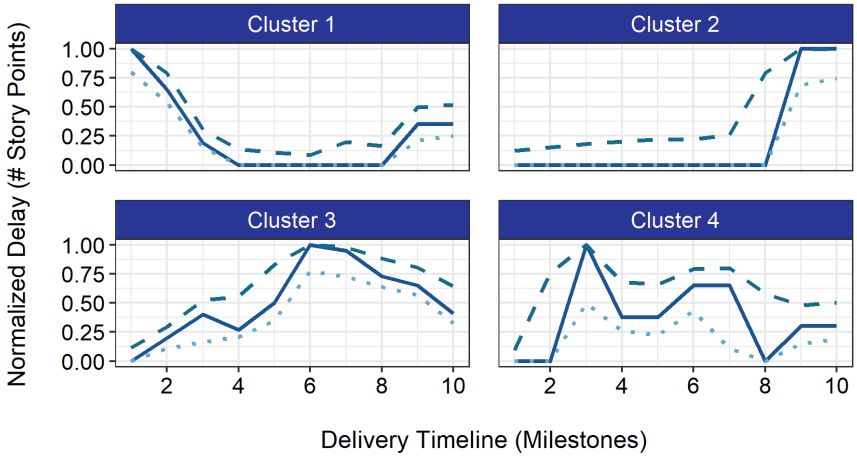


Figure 3.4: Four clusters of delay profiles representing recurrent delay patterns across milestones in epic deliveries at ING: 25th percentile: dotted; centroid: solid; and 75th percentile: dashed.

The clusters can be described as follows:

- *Cluster 1* (C1) consists of 1388 (36%) epics. These deliveries start out with a delay peak, followed by multi-phase recovery, and end with delay that continues beyond the planned delivery date. The epics of C1 have a significantly higher number of outgoing dependencies and developer workload, likely causing issues at the start of the delivery.

- *Cluster 2* (C2) makes up the largest group, containing 1706 (44%) epics that are punctual up until the last few milestones. The epics of C2 have a significantly higher security level and team stability, possibly explaining the consistent start. They also run into more incidents and unplanned work, likely causing the delay at the end of the delivery.

- *Cluster 3* (C3) contains 540 (14%) epics that exhibit an upward trend (i.e., delay increase) in the first section of the delivery followed by resilient recovery. The epics of C3 involve significantly smaller teams, suggesting that teams with fewer members may need some buildup time to respond to delay.

- *Cluster 4* (C4) contains 232 (6%) epics that exhibit a fluctuating pattern of delay increase and recovery over the course of the delivery. The epics of C4 have a significantly higher stability and lower security level, developer workload and number of sprints. These characteristics might possibly explain the consistent recovery of delay over time.

Table 3.2: Characteristics of delay profile clusters: Cluster 1 (C1), Cluster 2 (C2), Cluster 3 (C3), Cluster 4 (C4). * indicates that a cluster is significantly different from all other clusters for the corresponding predictor variable (pairwise Wilcoxon tests with Bonferroni correction).

Predictor	**Median**				**Significance**			
	C1	C2	C3	C4	C1	C2	C3	C4
nr-sprints	13	15	14	11				*
out-degree	7	3	4	4	*			
hist-performance	0.69	0.67	0.74	0.61				
dev-age-ing	2.49	2.61	2.92	2.84				
team-existence	1.30	1.53	1.29	1.42				
team-size	8	7	6	7			*	
security-level	0.56	0.77	0.53	0.36		*		*
unplanned-stories	0.11	0.16	0.10	0.08		*		
changed-leads	3	2	3	2				*
stability-ratio	0.73	0.81	0.64	0.72		*		
nr-stories	52	43	39	45	*			
nr-incidents	8	12	8	6		*		
dev-workload	15	12	10	8	*			*
BRE	0.23	0.17	0.11	0.09	*	*	*	*

Patterns are indicative of overall delay. The bottom row of Table 3.2 provides the descriptive statistics of the overall delay, measured in BRE values, for each cluster. The epics assigned to Cluster 1 suffer the largest overall delay with a median BRE of 0.23. The epics in Cluster 2 are associated with the second largest overall delay (median BRE of 0.17). Clusters 3 and 4 consist of epics that end up with small overall delays, with a median BRE of 0.11 and 0.09, respectively. Using the the Wilcoxon test for pairwise comparisons, we found that the differences in the BRE values of the clusters are statistically significant at a 95% confidence level. This means that the patterns are indicative of the overall epic delay.

3.5 Evaluation

3.5.1 Research Questions

The evaluation of the dynamic model aimed to answer the following research questions:

- **RQ1. Benefits of dynamic prediction:** *Does the dynamic model provide more accurate estimates than its global and global iterative modes?* To study the benefits of the proposed dynamic model, we evaluate the performance of the Bayesian model in global, global iterative and dynamic settings.
- **RQ2. Benefits of delay patterns:** *Does the use of delay patterns have a positive impact on the predictive performance?* We compare the performance of the dynamic Bayesian model learned with and without the delay patterns.

– **RQ3. Comparison with SoTA baselines:** *How does our dynamic Bayesian model compare to the state-of-the-art baselines?* To determine whether our dynamic Bayesian model improves the state-of-the-art (SoTA) baselines in effort estimation, we compare it with the Decision Tree model of Choetkiertikul et al. [183] and the Random Forests model of Choetkiertikul et al. [22]. We perform the comparison with the models in their original, global mode using features from Choetkiertikul et al. and in dynamic mode using our set of features.

– **RQ4. Impact of prediction time:** *How does the moment of prediction affect the informativeness of the predictions of the dynamic model?* Previous work [200, 201] has shown that statistical models should be evaluated in terms of both accuracy and informativeness (i.e., width of the prediction interval). We analyze how the informativeness of the predictions of the dynamic model evolves with the time of prediction (early versus late in the epic).

3.5.2 SoTA Baselines

We implemented two models representing the SoTA baselines in their original, global mode and dynamic mode for comparison with our dynamic Bayesian model. For comparison in global mode, we implemented the global Decision Tree model of Choetkiertikul et al. [183] using the five issue-level features presented in the paper. We mapped the features to the epic-level and extracted them from ING data. An overview of all variables and their mapping to the epic-level can be found in an online replication package [120]. We also implemented the global iterative Random Forests model of Choetkiertikul et al. [22] using 16 out of 19 features from the paper. We were not able to extract the variables 'number of fix versions', 'changing of fix versions' and 'number of affect versions' as they are specific to the context of issue reports. We converted the features to the epic-level, as described in the replication package. For comparison in dynamic mode, we implemented both models of Choetkiertikul et al. following the dynamic setup described in Section 3.3.4. The models were learned using our features from Table 3.1 and the delay patterns.

3.5.3 Experimental Setup

We performed experiments on the 4,040 epics in the ING dataset. To mimic a real prediction scenario, in which observed epics are used to inform predictions for future epics, we sorted the epics and their milestones based on their start date. For training and evaluation, we used time-based 10-fold cross-validation. The time-based variant of cross-validation ensures that in the k-th split, the epics in the first k folds (training set) are created before the epics in the (k+1)th fold (test set). The successive training sets are thus supersets of previous ones. This allows for the sequential updating of models based on past knowledge.

The Bayesian model estimates a probability distribution of BRE values. For evaluation, we selected the median of the posterior distribution as the predicted BRE value. This is a common approach when the goal of the model is to minimize the absolute or relative estimation error [201]. For the SoTA baselines, we applied the Decision Tree and Random Forests regressors to obtain a BRE estimate.

3.5.4 Performance Measures

We used the *Mean Absolute Error* (MAE) and the *Standardized Accuracy* (SA) as error measures; both have been recommended to compare the performance of effort estimation models [62, 202]. MAE is defined as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} | \, Actual\ BRE_i - Estimated\ BRE_i \, |$$

where N is the number of epics used for evaluation, $Actual\ BRE_i$ is the actual delay measured in BRE, and $Estimated\ BRE_i$ is the predicted BRE value, for an epic i. SA is based on MAE and compares an effort estimation model against random guessing:

$$SA = \left(1 - \frac{MAE}{MAE_{rg}}\right) \times 100$$

where MAE is defined as the MAE of the model that is being evaluated and MAE_{rg} is the MAE of a large number of random guesses. SA represents how much better the model performs than random guessing. We used the unbiased exact calculation of MAE_{rg} as proposed by Langdon et al. [202]. A lower MAE and higher SA imply better predictive performance.

To evaluate informativeness of the predictions of the Bayesian model (RQ4), we measured the relative width ($RWidth_{90}$) of the 90% credible intervals [200]. A narrower interval (i.e. lower $RWidth_{90}$) is more informative.

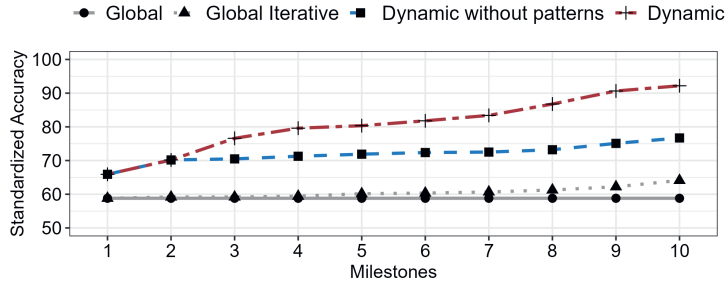
To compare model performance, we tested the statistical significance of the evaluation results using the Wilcoxon Signed Rank Test [191]. We applied the non-parametric Vargha and Delaney's $\hat{A}_{12}$ statistic [191], which is commonly used as effect size measure in effort estimation [62].

3.5.5 Results

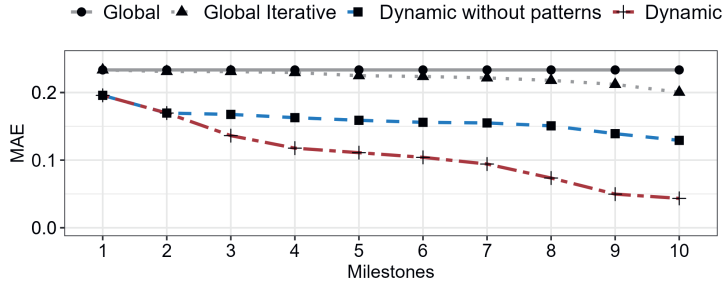
RQ1: Benefits of dynamic prediction. Figure 3.5 presents the evaluation results of the global, global iterative and dynamic modes of the Bayesian model for predicting the overall delay (in BRE) over milestones. Averaging across epics, the dynamic mode achieves 66–92% SA and 0.19–0.04 MAE over milestones. Over time, the dynamic mode consistently outperforms the global mode by 12–57% (SA) and 16–81% (MAE), and the global iterative mode by 12–44% (SA) and 16–78% (MAE). The Wilcoxon test shows that the improvements achieved by the dynamic mode are significant ($p < 0.001$) with medium to large effect sizes ($\hat{A}_{12} = [0.65, 0.81]$). *This indicates that the dynamic mode significantly improves global and global iterative modes right from the first milestone on.*

RQ2: Benefits of delay patterns. The dashed lines in Figure 3.5 present the evaluation results of the dynamic Bayesian model learned with and without delay patterns as input feature. At the first two milestones, the dynamic model learned using patterns provides the same estimations as the dynamic model learned without patterns. This is caused by the fact that the pattern clustering label becomes available from the third milestone on (i.e., when there is a series of two or more previous milestones to classify). Then, from

the third milestone on, the dynamic model learned using patterns consistently improves the dynamic model without patterns by 9–20% (SA) and 19–66% (MAE). The improvements achieved by using delay patterns are significant with medium effect size ($\hat{A}_{12} = [0.64, 0.69]$). *This indicates that the use of delay patterns leads to significant improvements in predictive performance, from the third milestone on.*



(a) Standardized Accuracy over time



(b) Mean Absolute Error over time

Figure 3.5: Evaluation results obtained by the global, global iterative and dynamic Bayesian models over milestones (RQ1); dynamic with and without delay patterns (RQ2).

RQ3: Comparison with SoTA baselines. Figure 3.6 presents the results of our dynamic Bayesian model compared to the SoTA baselines, represented by the Decision Tree [183] and Random Forests [22] models, in global and dynamic modes. The solid lines show the results of the Decision Tree and Random Forests models in their original, global mode using features from Choetkiertikul et al. [22, 183]. The dashed lines show the results of the models in dynamic mode using our features from Table 3.1.

The dynamic Bayesian model consistently outperforms the SoTA baselines in both global and dynamic modes. Over time, dynamic Bayesian improves the global Decision Tree by 74–144% (SA) and 44–87% (MAE), and the global iterative Random Forests by 56–71% (SA) and 34–84% (MAE). The Wilcoxon test shows that the improvements achieved by dynamic Bayesian over the global SoTA baselines are significant with large effect size ($\hat{A}_{12} > 0.84$). Dynamic Bayesian also outperforms the dynamic Decision Tree by 22-48% (SA) and 26–80% (MAE), and the dynamic Random Forests by 4–20% (SA) and 7–68% (MAE)

over milestones. The improvements of dynamic Bayesian over the dynamic Decision Tree and Random Forests are significant with effect sizes greater than 0.58. *This indicates that the dynamic Bayesian model achieves significant improvements over the SoTA baselines.*

Overall, the models in dynamic mode substantially outperform their counterparts in global mode. This highlights the benefits of dynamic predictions across models. *Bayesian achieves the highest predictive accuracy and the largest overall increase in performance compared to the SoTA baselines.*

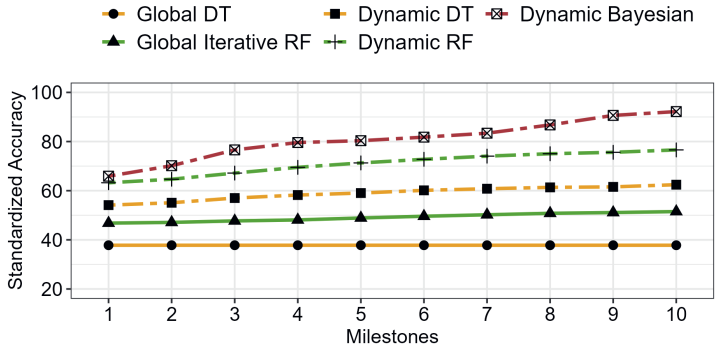


Figure 3.6: Comparison of our dynamic Bayesian model with SoTA baselines in global and dynamic modes (RQ3). 'Global DT' and 'Global Iterative RF' are the global Decision Tree [183] and global iterative Random Forests [22] learned using features from related work. 'Dynamic DT' and 'Dynamic RF' are the dynamic Decision Tree and dynamic Random Forests learned using our features from Table 3.1.

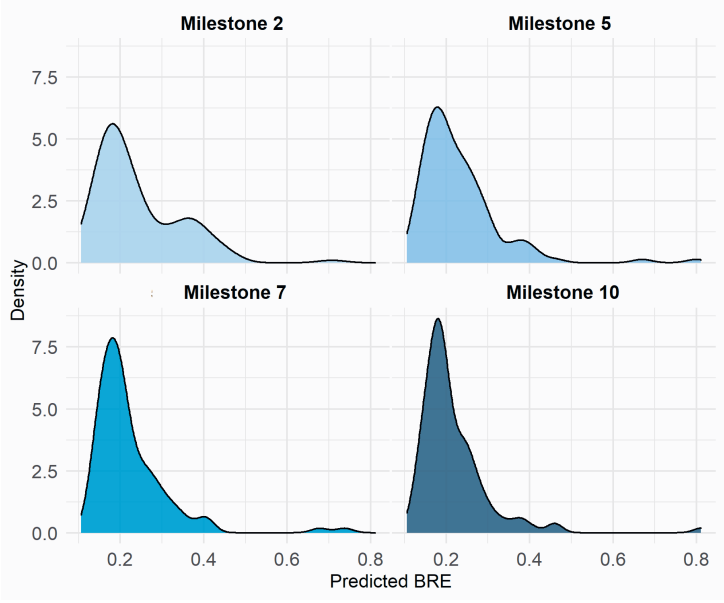


Figure 3.7: The estimated BRE distributions as updated by the dynamic Bayesian model across milestones (RQ4).

RQ4: Impact of prediction time. Figure 3.7 shows how the estimated BRE distributions of the dynamic Bayesian model evolve over milestones 2, 5, 7 and 10. The prediction intervals become more narrow and sharp over time. The average $RWidth_{90}$ of the prediction intervals decreases from 1.14 at milestone 2 to 1.01 at milestone 5, 0.94 at milestone 7, and 0.89 at milestone 10. The Wilcoxon test shows that the changes in $RWidth_{90}$ over time are significant ($p < 0.001$) and the effect sizes are small to medium ($\hat{A}_{12} = [0.59, 0.68]$). *This indicates that the dynamic Bayesian model is convergent, i.e. the predictions of the model become more certain and informative over time.*

3.6 Discussion

3.6.1 Main Findings

Delay patterns as input feature. We found that the patterns identified at the case company are indicative of the overall project delay. This means that a similar pattern in historical data can provide an estimation for the future development of delay in an ongoing project. The patterns have shown their value in transport, and now in software development as well. They can be useful as input feature for delay prediction and rescheduling decisions. Our results demonstrate that the use of patterns leads to significant improvements of 9–20% (SA) and 19–66% (MAE) in the predictions of delay. The patterns in other organizations might differ from the four patterns identified at the case company. We expect that the number and shape of patterns will depend on the dataset being used. The patterns are essentially a reflection of recurring problems or abnormal behaviors that lead to delay in organizations.

Relationships with risk factors. We characterized the patterns in terms of risk factors, as shown in Table 3.2. Our statistical analysis reveals that the patterns show significant differences in various risk factors. Even though we cannot reason about causal links between the factors and patterns, the results of our factor analysis enable us to form hypotheses on the causes of delays. For example, the epics in Cluster 1 have a significantly higher number of outgoing dependencies, larger delivery scope and higher developer workload. We therefore hypothesize that large epics with many dependencies and overloaded developers are likely to exhibit a pattern similar to that of Cluster 1 and lead to major overall delay. Testing such hypotheses could lead to actionable insights and suggest delay mitigation measures. For a comprehensive view, we recommend the use of both epic- and story-level risk factors to characterize the patterns. Epic-level risks can provide high-level insights into problems related to the environment that the delivery takes place in. Story-level risks can give lower-level insights into problematic software tasks and collaboration challenges (e.g., user stories that have an abnormal waiting time are an indication of lack of team cooperation [183]).

Benefits of dynamic prediction. Our results show that dynamic models significantly outperform their global and global iterative counterparts. The dynamic Bayesian model achieves improvements of at least 12–44% (SA) and 16–78% (MAE) right from the first milestone on. It also substantially outperforms the SoTA baselines. This highlights the benefits of dynamic prediction methods and indicates that existing, static methods are less

suited to predict long-term delay. Existing models are not able to adequately incorporate changes occurring during project execution. Dynamic methods can effectively incorporate dynamic phenomena, resulting in increasingly more accurate and reliable schedule estimates over time. Dynamic prediction can therefore help teams detect risks throughout the project life cycle and react to delays in a more prudent fashion. This is especially valuable in development settings that are subject to constant change and where schedule overruns are a critical factor.

Trade-off between prediction time and accuracy. Our evaluation results show that the predictions of the dynamic model become more accurate and informative over time. We acknowledge that predicting at later times (at 70—100% of the planned duration) may be less useful as it might be too late to change the outcome. However, the increased certainty may justify mitigation actions focused on handling a certain delay (e.g., postpone product launch, move features to other epic) instead of trying to catch up (by adding more resources). Furthermore, the dynamic approach achieves meaningful improvements right from the start of the project on. It improves the global and global iterative approaches by 12% (SA) and 16% (MAE) at 10% duration, 19% (SA) and 27% (MAE) at 20% duration and 29% (SA) and 41% (MAE) at 30% duration. The improvements add up to 34% (SA) and 37% (MAE) at 50% duration. The improvements obtained during the first half of the project are notable and can enable teams to take early measures against delay.

Benefits of Bayesian methods. In our comparison of the SoTA baselines in dynamic mode, Bayesian performs better than the Decision Tree and Random Forests models. Bayesian also achieves the largest overall increase in accuracy over time. This suggests that Bayesian is more effective in quantifying and updating the uncertainty of predictions over time. The results of RQ4 confirm this observation: the predictions of the Bayesian model become substantially more certain and informative over time. Unlike the other models, Bayesian provides detailed information about the uncertainty of an estimate in the form of a probability distribution. This can help organizations raise confidence in project plans.

3.6.2 Future Work

Causal inference. To improve the implementation of delay countermeasures, there is a need to better understand the causes of delays and delay patterns. An interesting direction for future research is to investigate why risk factors and delay patterns are related. This could be assessed by causal inference on individual patterns. Causal discovery (e.g., [203]) could be used to learn a causal graph from the time series and identify the underlying causes of trends or fluctuations in the patterns. This can help software organizations to identify the causes of specific delays and estimate the effects of corrective actions beforehand. Another opportunity for future work is to map recurring peak moments in patterns onto development activities to identify key drivers of delay. Initial work in this direction has been carried out by Kerzazi and Khomh [204] and Kula et al. [205]. Both studies found that testing is one of the most time consuming activities and likely to result in delay.

Systematic patterns. The identified delay patterns might be affected by systematic effects that are calendar-related. Previous work (e.g., [70, 206]) has shown the existence of such effects in software development work. An interesting opportunity for future research is to test for seasonality and model the time dependency of delay patterns using pattern matching. This would allow generalization over delay patterns and support the identification of systematic effects at different levels of time granularity. For instance, within-week dynamics due to day-of-the-week effects, and within-year dynamics affected by seasonal effects.

Event-driven prediction. Previous studies (e.g., [129, 161]) have found that software deliveries can be delayed by disruptive events, such as bugs and live incidents, that occur during project execution. Existing effort estimation models are static and therefore not able to incorporate such events into their predictions. Our dynamic model provides future research an opportunity to process incoming incidents as they occur. This would require updating of the model every time an incident or other notable event occurs. Previous studies [207, 208] have recognized the potential of event-driven models for improving replanning strategies in software projects.

Delay propagation. Currently, our dynamic model considers each software delivery independently and does not capture the interactions between dependent deliveries. However, a single delayed software delivery may cause a domino effect of secondary delays over dependent teams and projects. Future work should model the (dynamic) interrelation and propagation of delays across software deliveries. This could lead to more accurate estimates and a better understanding of the effects of delay propagation on delay patterns. Initial work in this direction has been carried out by Choetkiertikul et al. [21]. They have shown that the use of networked data and collective classification leads to significant accuracy improvements.

3.7 Threats to Validity

Construct validity. The data variables we consider may not capture the intended meaning of (concepts affecting) delay. This introduces possible threats to construct validity [176]. The delay measurements are derived from delivery dates and reported story points in the backlog management data. However, it might happen that teams do not take their delivery deadlines seriously and close their deliveries too early or too late. It is also possible that some teams do not follow the guidelines or principles for estimating story points. We tried to mitigate these threats by collecting real-world data from many epics and teams over a five year span.

Another potential threat to our study is related to the milestone division of epics. We split the epics into regularly-spaced milestones based on completion rate. However, the milestones may not be a good match with the work pace of some teams. This might have led to a mixture of project phases within milestones and across epics, which would affect the results for the patterns in some deliveries. In practice, it would be more appropriate to split the epics based on iterations.

Internal validity. The delay patterns that we condition our Bayesian model on may not reflect the situation in the test data. To mitigate this problem, we used time-based cross-validation to mimic a real prediction scenario. To compare models and verify our findings, we selected unbiased error measures and applied statistical tests [39, 191].

External validity. External threats are concerned with our ability to generalize our results. We have analyzed 4,040 epics from 270 teams, which differ significantly in size, composition and product domains. However, we acknowledge that our data may not be representative of software projects in other organizations and open source settings. In other contexts, software deliveries might have a different setup following different collaboration practices. Replication of our work is needed to validate the findings in other settings and reach more general conclusions.

3.8 Conclusions

Modern agile software projects are volatile due to their iterative and team-oriented nature. Changes in risk factors and team performance trigger the need to re-assess overall delay risk throughout the project life cycle. Existing effort estimation models are static and not able to capture changes occurring during project execution. In this paper, we have proposed a dynamic effort estimation model for continuously predicting overall delay using delay patterns and Bayesian modeling. The model incorporates the context of the project phase and is finetuned based on changes in delivery performance over time. We apply our approach to real-world data from thousands of epics, identifying four intuitive delay patterns at ING. The evaluation results demonstrate that:

1. Delay patterns are indicative of the overall delay and useful as input feature for dynamic prediction.
2. The dynamic model consistently outperforms global and global iterative approaches, and the SoTA baselines, even during early milestones (10–30% of project duration).
3. The predictions of the dynamic Bayesian model become substantially more certain and accurate over time.

Overall, our results highlight the benefits of dynamic prediction methods that are able to learn from the time-dependent characteristics of software project delays. We identified several research areas calling for further attention, including causal inference, systematic effects, and delay propagation. Progress in these areas is crucial to better understand and manage delays in software projects.

4

Modeling Team Dynamics for the Characterization and Prediction of Delays in User Stories

In agile software development, proper team structures and effort estimates are crucial to ensure the on-time delivery of software projects. Delivery performance can vary due to the influence of changes in teams, resulting in team dynamics that remain largely unexplored. We explore the effects of various aspects of teamwork on delays in software deliveries. We conducted a case study at ING and analyzed historical log data from 765,200 user stories and 571 teams to identify team factors characterizing delayed user stories. Based on these factors, we built models to predict the likelihood and duration of delays in user stories. The evaluation results show that the use of team-related features leads to a significant improvement in the predictions of delay, achieving on average 74%-82% precision, 78%-86% recall and 76%-84% F-measure. Moreover, our results show that team-related features can help improve the prediction of delay likelihood, while delay duration can be explained exclusively using them. Finally, training on recent user stories using a sliding window setting improves the predictive performance; our predictive models perform significantly better for teams that have been stable. Overall, our results indicate that planning in agile development settings can be significantly improved by incorporating team-related information and incremental learning methods into analysis/predictive models.

This chapter has been published as *E. Kula, A. van Deursen, and G. Gousios. Modeling Team Dynamics for the Characterization and Prediction of Delays in User Stories, IEEE/ACM ASE'21 [131].*

The overall perceived success of a software project depends heavily on the timeliness of its delivery [9]. Reducing delays is therefore a critical goal for software companies. Over the past two decades, software organizations have increasingly embraced agile development methods to manage software projects [209]. Agile gained popularity in the software industry because, in comparison to traditional (waterfall-like) approaches, it uses an iterative approach to software development, aimed at reducing development time, managing changing priorities and inherently reducing risk [11]. However, on-time delivery remains a challenge in agile software development. Prior work [12] has found that around half of the agile projects run into effort overruns of 25% or more.

In agile settings, software is incrementally developed through short iterations to enable a fast response to changing markets and customer demands. Each iteration requires the completion of a number of *user stories*, which are a common way for agile teams to express user requirements. Agile teams are responsible for determining the next iteration's workload together and then breaking these into user stories that can be implemented, tested and shipped in one iteration. Agile teams are characterized by self-organization and intense collaboration [11, 210]. Several studies [9, 104, 211–213] have shown the importance of teamwork for the success of agile projects. Various aspects of teamwork, such as team orientation, team coordination and work division, can affect software delivery performance [104, 105]. Moreover, delivery performance can vary due to the influence of changes in teams, resulting in team dynamics that remain largely unexplored. Hence, there is a need to better understand the effects of teamwork and team dynamics on delays, which can benefit the effective application of agile methods in software development.

Today's agile projects require different approaches to planning due to their iterative and team-oriented nature [13]. Central to the planning is the ability to predict, at any phase of the project, if a team can deliver the planned software features on-time. Agile teams would therefore benefit from team-specific, actionable information about the current existence of delay risks at the fine-grained level of user stories, allowing them to take measures to reduce the chance of delays. Recent approaches have leveraged machine learning techniques for evaluating risk factors in software projects (e.g., [98, 100]), estimating effort for issue reports (e.g., [18–20]) and predicting delays in bugs or issues (e.g., [21–23]). These approaches focus on the technical aspects of software deliveries and do not adequately take into account team-related factors. Studies of software teams [104–106, 214] have developed theoretical concepts and detailed performance models that articulate relationships between various aspects of teamwork quality and the extent to which a team is able to meet time and cost objectives in software projects. These studies point out that various aspects of teamwork need to be considered when planning software deliveries. Therefore, the predictive power of existing effort prediction models might be enhanced by incorporating such factors.

In this paper, we explore the effects of various aspects of teamwork on delays in software deliveries. Project delays are common in the software industry [4, 12], which makes it important to study this phenomenon in more detail. There is a need to understand and predict, especially during early project phases, which projects will be delayed. This would allow teams to better manage and possibly prevent delays. To do so, we conduct a case study at ING, where around one quarter of its user stories are delayed. We analyze historical log data from 571 teams and 765,200 user stories at ING to identify team factors

characterizing delayed user stories. Based on these factors, we build models that can effectively predict the likelihood and duration of delays in user stories. Our models learn from a team's past delivery performance to predict delay risks in new user stories. To determine whether the use of team features has a positive impact on the predictive performance, we compare the results of models learned using different sets of features: story features, text features and team features. We also evaluate the models with a sliding window setting to explore incremental learning and the impact of team churn on the models' performance. The sliding window works as a forgetting mechanism: the model learns from a team's recent delivery performance in the window and forgets older, irrelevant data to follow team changes over time.

Our results show that the use of team features leads to a significant improvement in the predictions of delay, achieving on average 74%-82% precision, 78%-86% recall, 76%-84% F-measure and 80%-92% AUC. Team features can help improve the prediction of delay likelihood, while delay duration can be predicted exclusively using them. Moreover, developer workload, team experience, team stability and past effort estimates are the most important team features for predicting delay. Finally, training on recent user stories using a sliding window improves the predictive performance; our predictive models perform significantly better for teams that have been stable.

4.1 Usage Scenarios

At the end of an iteration, a number of user stories are completed and there may also be a number of incomplete/unresolved user stories delayed to future iterations. Our prediction models enable teams to identify these user stories before the start of an iteration. There are two scenarios in which predictions are being made: *before* and *after* an effort estimate has been made for a user story. The availability of an estimate might affect the accuracy and usefulness of our predictions. It is likely that in the latter scenario, our predictions get more accurate (since we have information about the estimated size of a user story) but the less useful it is (since the team has already spent a considerable amount of time on estimating the story).

In both scenarios, our predictive models can be used as a decision support system to generate proactive feedback and make informed decisions on the planning and feasibility of a user story. Foreseeing delay risks allows teams to identify problematic user stories and take corrective actions, such as story splicing (i.e., splitting large stories into smaller ones) or resolving inter-story dependencies. Our models learn from the past delivery performance of the specific team which they are deployed to assist. Hence, the predictions our models make are team-specific. This helps teams improve their schedule estimates and gain an increased awareness of their own behavior patterns.

4.2 Study Design

In this paper, we propose a team-driven approach to determine early on the impact and probability of a delay risk occurring in a user story. To do so, we extract 24 risk factors representing technical and team-related aspects of a user story. Based on these factors, we build models that can effectively predict the likelihood and duration of delays in user stories. Predictions are generated for two usage scenarios: before and after an effort esti-

mate has been made for the user story. For convenience, in the remainder of this paper, we denote the scenarios as SC1 and SC2, respectively.

Throughout our study, the following research questions guide our work:

- **RQ1. Benefits of team features:** *Does the use of team features have a positive impact on our predictive performance? (RQ1.1), How effective is our approach when team features are used exclusively? (RQ1.2)* To answer these questions, we compare the performance of models learned using combinations of different sets of features: story features, text features and team features.
- **RQ2. Feature importance:** *Which team features are most important for predicting delays in user stories?* For this question, we train models using the extracted 24 features and determine the relative importance of team features in terms of predictive power.
- **RQ3. Benefits of sliding window:** *Does the use of a sliding window provide more accurate and robust estimates?* As teams change over time and these changes might affect their delivery performance, we want to analyze whether it is beneficial for our predictive model to learn from recent user stories and forget older data. To do so, we compare the performance of models learned using all features in a sliding window setting versus expanding window setting.
- **RQ4. Factor of change:** *How does team churn affect story delays and our predictive performance?* Changes in team composition (due to either a member leaving or joining the team) can cause teams to become less predictable at delivering software. We employ the sliding window setting and determine for each user story the number of consecutive windows a team has been stable for. We perform a statistical analysis to assess the impact of the number of consecutive stable windows on story delays and our models' performance.

We can split our approach into four main steps:

1. *Data collection and pre-processing:* We collect and pre-process backlog management data (past user stories) from 571 development teams at ING.
2. *Risk factor extraction and analysis:* We extract 24 risk factors representing technical and team-related aspects of user stories, and then perform correlation analysis to determine whether the factors affect delays in user stories.
3. *Text feature extraction:* We use RoBERTa [215], a state-of-the-art language representation model, to produce vector representations (i.e., embeddings) for the textual descriptions of user stories. To adapt the model to our prediction task, we update it with additional training on our corpus of unlabeled user stories.
4. *Model building:* We use the selected risk factors and text embeddings to build models that predict delays in user stories.
5. *Model evaluation:* We evaluate our models using various sets of features in different experimental settings to answer the research questions.

4.2.1 Data Collection and Pre-Processing

We extracted log data from *ServiceNow*, a backlog management tool used by a majority of teams at ING [216]. The dataset consists of user stories delivered by 571 teams at ING between January 01, 2016 and January 01, 2021. The user stories have significant variety in terms of the products developed, the size and application domain (banking applications, cloud software, software tools). The dataset contains the following fields for user stories: *Identification Number*, *Creation Date*, *Sprint Identification Number*, *Planned Start Date*, *Actual Start Date*, *Planned Delivery Date*, *Actual Delivery Date*, *Story Points* and the textual *Title* and *Description* fields. The *Planned Start Date* coincides with the start date of the sprint that the story was originally assigned to. We acknowledge that the planned start date of a user story might change before the sprint is started. Therefore, we consider only the planned start date as scheduled on the day that the development phase of a sprint is started. For each user story, the dataset contains the entire history of changes. This enabled us to track the number of sprints a user story was delayed for. We acknowledge that a team might decide to temporarily move a story back to the product backlog after a delay. Therefore, we calculate the delay duration based on the number of sprints a story has actually been part of.

To eliminate noise and missing values, we removed user stories with a status other than 'Completed'. We also filtered out user stories with empty *Planned Delivery Date*, *Actual Delivery Date*, *Story Points* and *Description* fields. Moreover, we deleted user stories that have not been assigned to a developer. We also removed stories that were added to a sprint during the development phase, because they are likely to be unstable and not accurately represent delay. We found a few user stories that had been delayed for an unusually long period of time (e.g., in some cases over 10 sprints). We removed such outliers that exceed two standard deviations from the mean delay duration of all user stories. The original dataset contained 889,014 user stories. After removing outliers and pre-processing the data, the final dataset decreased to 765,200 user stories from 571 teams. This dataset consists of 183,342 (24%) delayed and 581,858 (76%) non-delayed user stories.

Risk classes. The delayed stories in our dataset consist of 76,398 (42%) stories that were delayed for a single sprint, 61,052 (33%) stories that were delayed for two sprints, 34,821 (19%) stories that were delayed for three sprints and 11,071 (6%) stories that were delayed for more than three sprints. For our predictions, we choose to use four risk classes that reflect the degree of delay: *non-delayed*, *minor delay* (delay of one sprint), *medium delay* (delay of two sprints) and *major delay* (delay of three sprints or more). Since a small fraction of the user stories in our data were delayed for more than three sprints, we decided to merge this group with the user stories that were delayed for three sprints.

4.2.2 Risk Factor Extraction and Analysis

We extracted 24 risk factors from the collected log data to explore which factors characterize delayed user stories. Table 4.1 provides an overview and correlation analysis of these factors. The factors are divided in two groups: *story factors* and *team factors*. The story factors represent the inherent characteristics of user stories, such as its size, type and priority. The team-related factors represent characteristics of individual team members and the group as a whole.

Table 4.1: The 24 extracted risk factors representing the characteristics of a user story and development team. Correlation coefficients are based on Spearman's Correlation [217]: they measure the strength of the relationship between factors and the risk classes. Statistical significance with Holm correction [218] is indicated with * (p-value < 0.01) and ** (p-value < 0.001).

Category	Factor name	Description	Type	Correlation coefficient	
				Spearman's ρ	Interpretation
Story factors	dev-type	The development type (1. new feature, 2. bug fix or 3. improvement) of a story	Categorical	−0.19*	Weak
	priority	Does a story have a major priority to the customer?	Binary	−0.31**	Weak
	security	Whether a story is associated with a security-critical system	Binary	0.44**	Moderate
	out-degree	Number of outgoing dependencies of a story on other stories	Continuous	0.41**	Moderate
	sprint-duration	Planned duration of the sprint that a story was originally assigned to	Continuous	−0.26**	Weak
	planned-stories	Total number of user stories in the sprint that a story was originally assigned to	Continuous	0.22**	Weak
	planned-points	Total number of story points in the sprint that a story was originally assigned to	Continuous	0.13**	Weak
	initial-points	Number of story points initially estimated for a user story	Continuous	0.51**	Moderate
Team factors	team-size	Number of team members	Continuous	0.08*	Weak
	avg-story-size	Average number of story points that the team assigned to past stories	Continuous	0.46*	Moderate
	team-existence	Number of years the team has existed for	Continuous	−0.35**	Weak
	team-stability	Ratio of team members that did not change in the last six months	Continuous	−0.40**	Moderate
	po-stability	Did the product owner of the team stay the same in the last six months?	Binary	−0.29**	Weak

(Continued on next page)

(Continued from previous page)

Category	Factor name	Description	Type	Correlation coefficient	
				Spearman's ρ	Interpretation
	team-capacity-stories	Total number of user stories that have been completed by the team so far	Continuous	-0.28^{**}	Weak
	team-capacity-points	Total number of story points that have been completed by the team so far	Continuous	-0.26^{**}	Weak
	global-distance	The Global Distance Metric [148] measured across teams members	Continuous	0.17^{*}	Weak
	dev-seniority	The seniority rank of a developer at ING	Categorical	0.24^{**}	Weak
	dev-age-team	Number of years spent by a developer in the current team	Continuous	-0.28^{**}	Weak
	dev-age-project	Number of years spent by a developer in the current project	Continuous	-0.12^{**}	Weak
	dev-age-abc	Number of years spent by a developer at ING	Continuous	-0.43^{**}	Moderate
	dev-workload-stories	Number of user stories assigned to a developer in the current sprint	Continuous	0.53^{**}	Moderate
	dev-workload-points	Number of story points assigned to a developer in the current sprint	Continuous	0.49^{**}	Moderate
	dev-capacity-stories	Total number of user stories that have been completed by a developer so far	Continuous	-0.45^{**}	Moderate
	dev-capacity-points	Total number of story points that have been completed by a developer so far	Continuous	-0.41^{**}	Moderate

Story factors. Several story factors are extracted directly from the story's primitive attributes, which include dev-type and priority. Each story will be assigned a type and priority which indicate the nature and urgency of the task associated with implementing the story. Both factors have been shown to affect the delivery of a story in related work [101]. We extract security to determine whether a user story needs to go through a mandatory, resource-intensive security testing procedure at ING that might lead to delay. We extract the outgoing degree (out-degree) of dependencies of a story; this has been shown to predict delay in related work [21]. The remaining story factors are used to extract the size of a story and the sprint.

Team factors. Previous work (e.g. [219, 220]) has found that member turnover can lead to tacit knowledge loss, and thus may negatively affect team productivity. Therefore, we compute the stability of a team (team-stability) and that of its product owner (po-stability). We also measure team-existence to quantify the familiarity and maturity of a team; both have been shown to lead to better team interactions and project performance in related work (e.g., [221, 222]). Previous studies (e.g., [223, 224]) have shown that the interactions among team members are less effective in distributed teams. To quantify the distance between team members, we calculate global-distance based on the Global Distance Metric proposed in related work [148]. We calculate the metric for pair-wise combinations of team members and take the maximum value.

Developers' capabilities and experience can influence their contributions to projects (e.g., [220, 225, 226]). Thus, we compute dev-capacity-stories and dev-capacity-points to quantify the software delivery experience of the developer that a user story is assigned to. Similarly, we use team-capacity-stories and team-capacity-points to quantify the team's overall experience with software deliveries. Moreover, we extract the developers' seniority; the intuition here is that senior developers might more often be assigned to complex user stories that have a higher delay risk. ING employs the five-stage Dreyfus Model [150] to assess developers' expertise based on their software industry experience. We extract dev-age-team, dev-age-project and dev-age-abc to measure how long team members have been working within their teams, projects and overall at ING.

Related work has identified an inappropriate division of work as an important barrier to achieving team effectiveness [104]. Hence, we calculate dev-workload-stories and dev-workload-points. Finally, we extract team-size and avg-story-size as larger projects are associated with greater risk in literature (e.g., [83, 221, 227]).

4.2.3 Text Feature Extraction

The title and description of a user story can provide good features since they explain the nature and complexity of a story. To extract features from text, we combined the title and description of a user story into a single text document where the title is followed by the description. Our approach computes vector representations for these documents that are then used as features to predict delays in user stories. We tried different methods for text feature extraction: the traditional Bag-of-Words (with *TF-IDF* [228] and *Okapi BM25* [229]), the neural network-based *Doc2Vec* [230] and the state-of-the-art transformer-based language model *RoBERTa* [215]. In case of TF-IDF and BM25, we pre-processed the

texts by lower casing the words and removing punctuation and stop words. We compared and evaluated the methods on our prediction task and corpus of user stories. We found that RoBERTa outperforms the other methods on average by 11%-27% in precision, 5%-38% recall and 9%-33% F-measure. Therefore, we decided to use RoBERTa as part of our experimental setup for answering the research questions.

RoBERTa is an optimization based on Google's BERT [231], which is able to learn bidirectional word embeddings from texts. To adapt the model to the domain-specific vocabulary of user stories, we updated it with additional training on our corpus of unlabeled stories. In this updating procedure, we implemented the same masked language modeling strategy used in the pre-training of the original model, with a set of newly designated hyperparameters (*training steps: 60K, batch size: 64, optimizer: Adam, learning rate 3×10-5*).

4.2.4 Model Building

Our objective is to predict the probability of a delay occurring (i.e., delay likelihood) and a probability distribution over the aforementioned risk classes (i.e., delay duration in terms of the number of sprints overrun). Therefore, our predictive models should be able to provide probability estimates. We employ *binary classification* for predicting the likelihood of delay. We reduce the aforementioned risk classes into two binary classes: *delayed* and *non-delayed*. The delayed class covers the user stories that belong to the aforementioned *minor delay*, *medium delay* and *major delay* classes. For predicting the delay duration of delayed stories, we employ *multi-class classification* for the *minor delay*, *medium delay* and *major delay* classes.

We compared and evaluated four different classifiers that are able to provide class probabilities and that have been shown to be effective classifiers in risk prediction: Random Forests [232], AdaBoost [233], Multi-layer Perceptron [234] and Naive Bayes [235]. The results of the model comparison can be found in an online replication package [121]. Random Forests outperforms the other classifiers on both prediction tasks, on average by 2%-23% in precision, 9%-48% in recall and 4%-33% in terms of F-measure. Therefore, we chose to employ Random Forests (RF) [232] as part of our experimental setup.

Figure 4.1 shows the design of our pipeline of predicting delays in user stories: (i) extract numerical story features, (ii) extract numerical team features, (iii) produce document representations of user stories using RoBERTa, (iv) concatenate features and (iv) classification using Random Forests.

4.2.5 Model Evaluation

Evaluation setup. We performed experiments on the 765,200 user stories in our dataset. We built team-specific predictive models, meaning that our models are trained and tested on a dataset containing the past user stories from one specific team. Hence, we built two models for each team in the dataset: one for predicting delay likelihood and one for predicting delay duration.

To mimic a real prediction scenario where the delay of a given user story is estimated based on knowledge from previous stories, we sorted the stories based on their start date. Then, for training and evaluation in RQ1 and RQ2, we used time-based 10-fold cross-validation. Cross-validation is a well-known technique to prevent the classifier from over-fitting. The time-based variant of cross-validation ensures that in the kth split, the

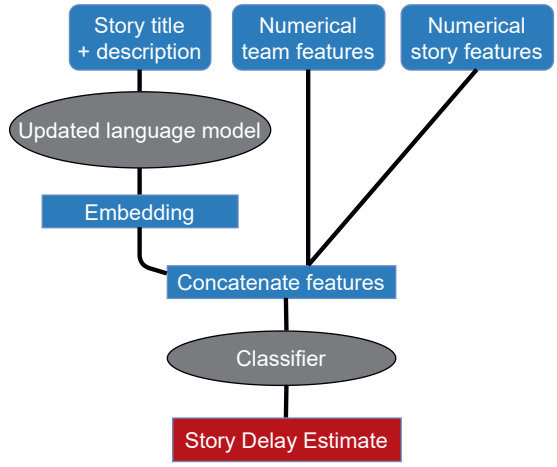


Figure 4.1: Our pipeline for predicting delays in user stories

stories in the first k folds (training set) are created before the stories in the (k+1)th fold (test set). Unlike standard cross-validation, successive training sets are supersets of previous ones (also known as an expanding window).

For RQ1, we evaluated the performance of the models learned using different sets of features. We ran all experiments for the two types of usage scenarios: SC1 (excluding the initial-points feature) and SC2 (including the initial-points feature). For RQ2—RQ4, we ran the experiments using all features (including initial-points).

Our predictive models are able to estimate class probabilities for the delay likelihood and delay duration of user stories. During the testing phase of our models, we chose the class with the highest probability as the predicted class.

Sliding window setting. For RQ3, we evaluated our predictive models using two different experimental settings: the expanding window and the sliding window. In both settings, the user stories are first sorted based on their start date and then divided into multiple time-based windows. For each window k_i in the expanding window setting, we use the stories from the previous windows $k_0 ... k_{i-1}$ to train a model. In the sliding window setting, however, we train the model only on the last window k_{i-1}. The sliding window allows us to train the model on a team's recent delivery performance, while the expanding window uses all observations available.

To analyze the impact of team churn on delays and our models' performance (RQ4), we employed the sliding window setting and determined for each window whether a team had been stable or not. We marked a team as stable during a window if no team members left or joined the team during that window (i.e., if the team composition did not change during that window). For each story, we determined the number of consecutive windows a team had been stable for. If the team composition had changed in the previous window, then this number was considered to be zero. Finally, we performed a statistical comparison of delays and our models' evaluation results between stable and unstable teams.

Performance measures. We computed the widely used precision, recall and F1-score to evaluate the performance of our predictive models. To account for class imbalance, we calculated the weighted averages of these measures (i.e, the score of each class is weighted by the number of samples from that class). We also used Area Under the Curve (AUC) of receiver operator characteristics (ROC) [236] in classifying the outcome of a user story.

To compare the performance of predictive models, we tested the statistical significance of their evaluation results using the Wilcoxon Signed Rank Test [191]. This is a non-parametric test that makes no assumptions about underlying data distributions. We employed the non-parametric effect size measure, the Vargha and Delaney's $\hat{A}_{12}$ statistic [191]. This measure is commonly used for evaluation in effort estimation [62].

4.3 Results

In this section, we report the results in answering research questions RQs 1-4.

4

RQ1: Benefits of Team Features

For this research question, we compared the performance of models learned using story features, a combination of story and text features, and all features (story, text and team features). We used the Wilcoxon test and $\hat{A}_{12}$ effect size to investigate whether the improvements achieved by adding team characteristics are statistically significant.

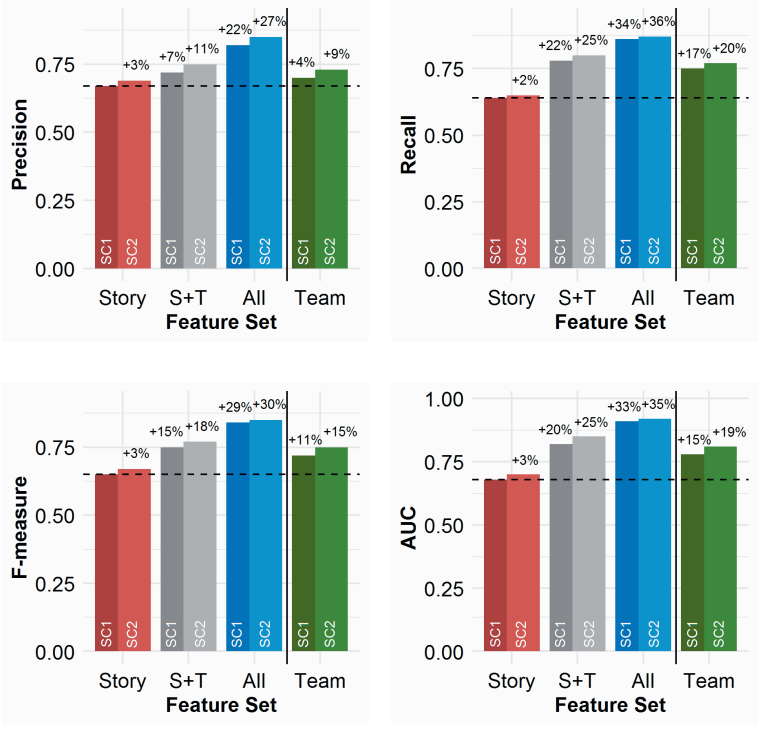


Figure 4.2: Results obtained for predicting delay likelihood in SC1/SC2: the *story features* baseline achieved 0.67/0.69 precision, 0.64/0.65 recall, 0.65/0.67 F1 and 0.68/0.70 AUC.

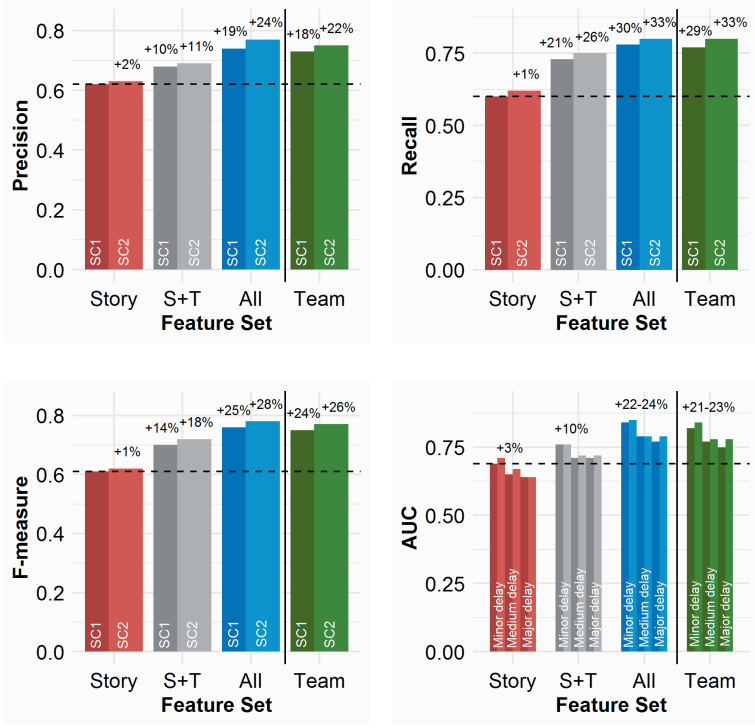


Figure 4.3: Results obtained for predicting delay duration in SC1/SC2: the *story features* baseline achieved 0.62/0.63 precision, 0.60/0.62 recall, 0.61/0.62 F1, 0.69/0.71 AUC (minor delay), 0.65/0.67 AUC (medium delay), and 0.64/0.64 AUC (major delay).

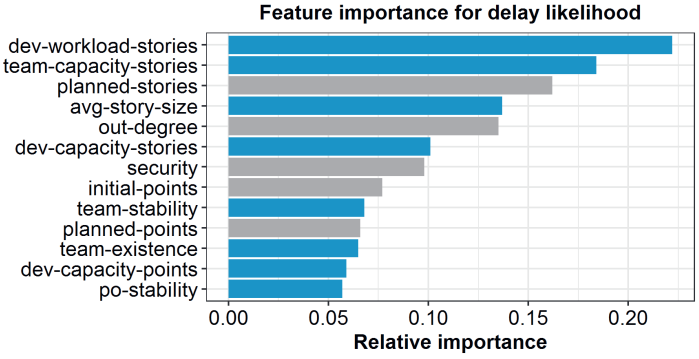
RQ1.1: Does the use of team features have a positive impact on our predictive performance? Figure 4.2 presents the evaluation results for predicting delay likelihood in usage scenarios SC1 and SC2 (described in Section 4.1). In both scenarios, the models learned using all features outperform the models learned using a subset of features in terms of precision, recall, F1 and AUC. On average, the all-features models improve the *story* and *story+text* models by 19% (precision), 23% (recall), 21% (F1-score) and 22% (AUC). Statistical tests show that the improvements achieved by the all-features models are significant ($p < 0.001$) and the effect sizes are large (ranging between 0.71 and 0.87). *This demonstrates that the addition of team features leads to a significant improvement in the predictions of delay likelihood.*

Figure 4.3 presents the evaluation results for predicting delay duration. Similarly, in both scenarios, the models learned using all features outperform the models learned using a subset of features in terms of precision, recall, F1 and AUC. On average, the all-features models improve the *story* and *story+text* models by 16% (precision), 20% (recall), 18% (F1-score) and 17% (AUC). These improvements are significant ($p < 0.001$) and the effect sizes are at least medium (ranging between 0.65 and 0.78). *This indicates that the addition of team features leads to a significant improvement in the predictions of delay duration.*

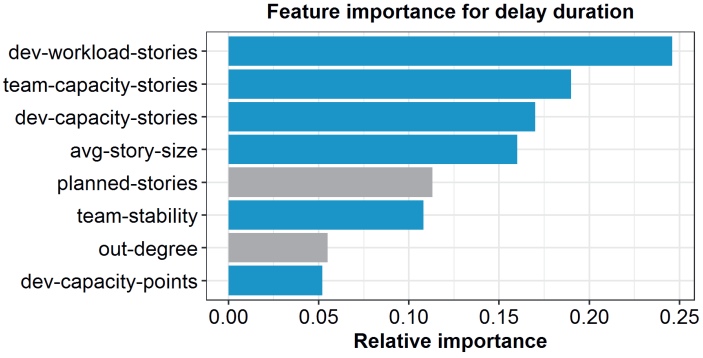
RQ1.2: How effective is our approach when team features are used exclusively? As shown in Figure 4.2, the models learned using team features only for predicting delay likelihood perform better than the *story* models and slightly worse than the *story+text* models. The improvements achieved by the *all-features* models over the *team* models are significant and the effect sizes are at least medium (ranging between 0.63 and 0.76). *This indicates that the three feature sets have statistically significant contributions to the predictions of delay likelihood.*

Figure 4.3 shows that the models learned using team features only for predicting delay duration achieve similar results as the *all-features* models. The statistical tests show that the differences between both models in terms of precision, recall, F1 and AUC are significant but the effect sizes are negligible. *This indicates that we can effectively predict delay duration using team features exclusively.*

RQ2: Feature Importance



(a)



(b)

Figure 4.4: Feature importance for predicting the likelihood and duration of delays in user stories: team features are highlighted in blue, story features in gray.

Using the feature importance evaluation built in Random Forests [237], we obtained the top most important features and their normalized weights from models learned using story

and team features (including initial-points). The text features were not included as it is not possible to reduce the vectors produced by RoBERTa to one single feature. The models learned using story and team features achieve on average 0.74/0.76 precision, 0.79/0.80 recall, 0.77/0.78 F1 and 0.81/0.81 AUC for predicting delay likelihood/duration. Figure 4.4 provides a ranking of the features by order of importance for predicting delay likelihood and delay duration. We averaged the importance values of the features across the teams in the dataset to produce an overall ranking. Features that have an importance value lower than 0.05 are not shown.

Figure 4.4a shows that 13 features from Table 4.1 contribute significantly to the predictions of delay likelihood. Dev-workload-stories, team-capacity-stories, planned-stories, avg-story-size and out-degree are the top-5 most important features. Their importance values range from 14% to 22%. Figure 4.4b shows that a partially overlapping set of 8 features is effective in predicting delay duration. Even though the top most important features in Figure 4.4b are similar to those for delay likelihood, there are a few ranking differences. Overall, the team features have greater explanatory power for delay duration than for delay likelihood. This corresponds to our results for RQ1. Dev-capacity-stories and team-stability play a significantly larger role in the predictions of delay duration.

RQ3: Benefits of Sliding Window

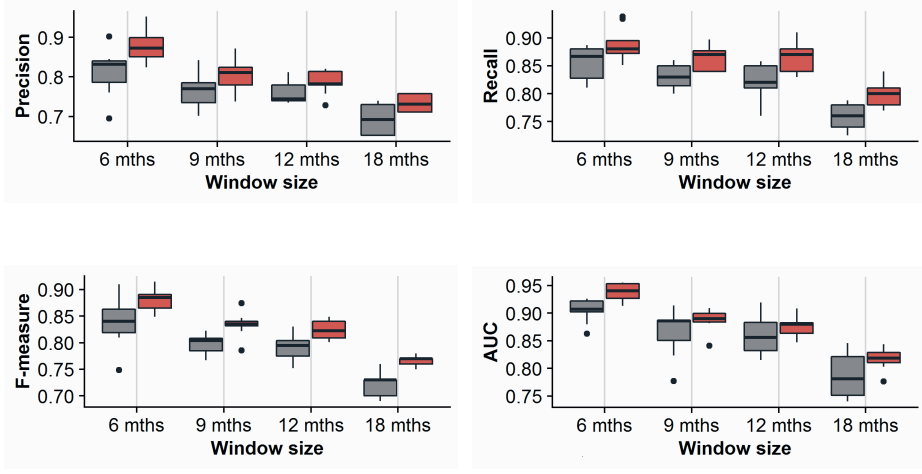
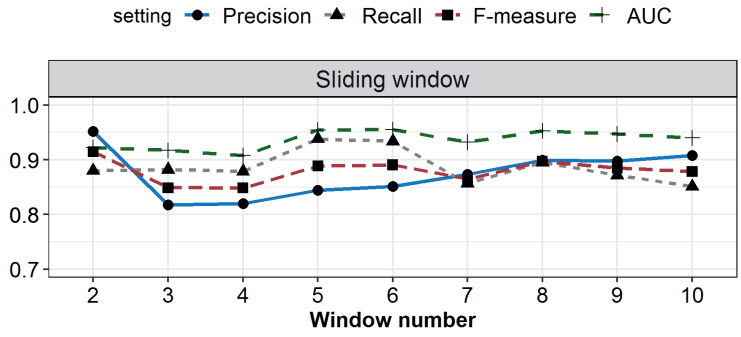


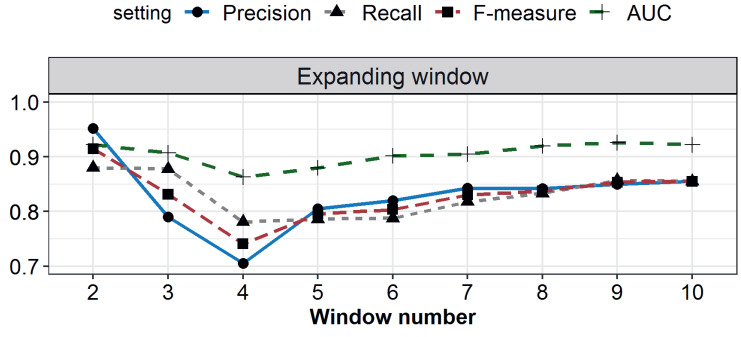
Figure 4.5: Evaluation results for predicting delay likelihood across different window sizes in sliding window (depicted in red) and expanding window (depicted in gray) settings.

Figure 4.5 presents the evaluation results obtained for predicting the likelihood of delay using an expanding versus sliding window. The results are averaged across windows and the teams in the dataset. As shown in Figure 4.5, the sliding window consistently outperforms the expanding window in terms of precision, recall, F1 and AUC across all window sizes. The Wilcoxon test shows that the improvements are significant ($p < 0.001$), and the effect sizes are between 0.55 and 0.68 (small to medium). Comparing the results across window

sizes, we observe that both the expanding window and the sliding window achieve the best performance for a window of six months.



(a)



(b)

Figure 4.6: Evaluation results obtained over time in the sliding window and expanding window settings (using a 6-month window).

Figures 4.6a and 4.6b visualize the evaluation results obtained over time in the sliding and expanding window settings using a 6-month window. The first window is not included as it is used for training only. Figure 4.6a shows lower variance over time in the prediction results for the sliding window. Both window settings start off with the same precision, recall, F1 and AUC scores for the second window and then their performance declines during the third and fourth windows. Further analysis of the data shows that a majority of teams have greater variance in their story delays during the initial windows. This might explain why the performance of both approaches decline at the start. We observe that the performance of the expanding window drops drastically during the initial windows, while the performance of the sliding window remains more stable. This suggests that the sliding window is better able to adapt to changes in teams' delivery performance.

RQ4: Factor of Change

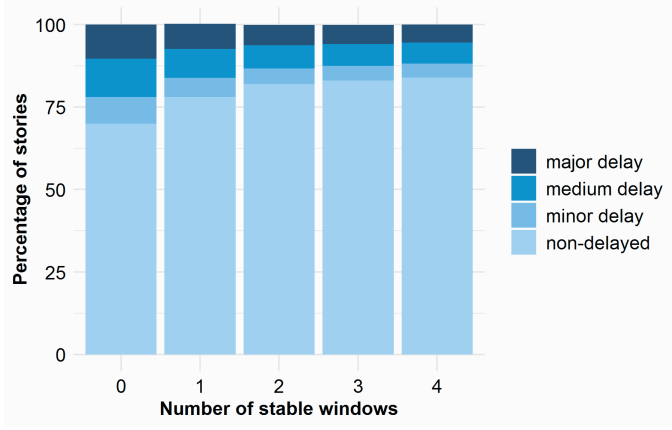


Figure 4.7: Delay percentage distribution across different stability levels

Figure 4.7 presents a percentage distribution of different levels of team stability based on the percentage of stories that were delayed. We observe that user stories that are delivered by stable teams are less likely to be delayed. 29% of the stories that have been delivered after a team change (i.e., zero stable windows) are delayed, of which 10% are hindered by a major delay. The percentages of delayed user stories decrease for teams that have been stable for a longer period of time. 21% of the stories that have been delivered after one stable window are delayed, of which 7% has a major delay. Only 15%-17% of the stories that have been delivered after more than one stable window are delayed.

Figure 4.8 presents the evaluation results obtained for predicting the likelihood of delay for stories delivered by teams of varying stability. We observe that our predictive models achieve better precision, recall, F1 and AUC scores for teams that have been stable. On average, the models achieve 19% higher precision, 13% higher recall, 16% higher F1 and 10% higher AUC for stories that are delivered after at least one stable window. Statistical tests show that these improvements are significant ($p < 0.001$) and the effect sizes are between 0.63 and 0.71 (small to medium). Figure 4.8 also shows a greater variance in the results for stories delivered after a team change.

4.4 Discussion

4.4.1 Recommendations for Practitioners

Our study provides practitioners with an extensive list of risk factors. By collecting and analyzing these factors, software companies can identify delay risks and derive useful models to predict delays in deliveries. Our models are effective in predicting the likelihood and duration of delays in user stories, achieving on average 74%-82% precision, 78%-86% recall, 76%-84% F-measure and 80%-92% AUC. Our models enable development teams to foresee, either before or after effort estimation, if a user story is at risk of being moved to a future iteration. This allows teams to identify problematic user stories and take corrective actions to reduce the chance of delays.

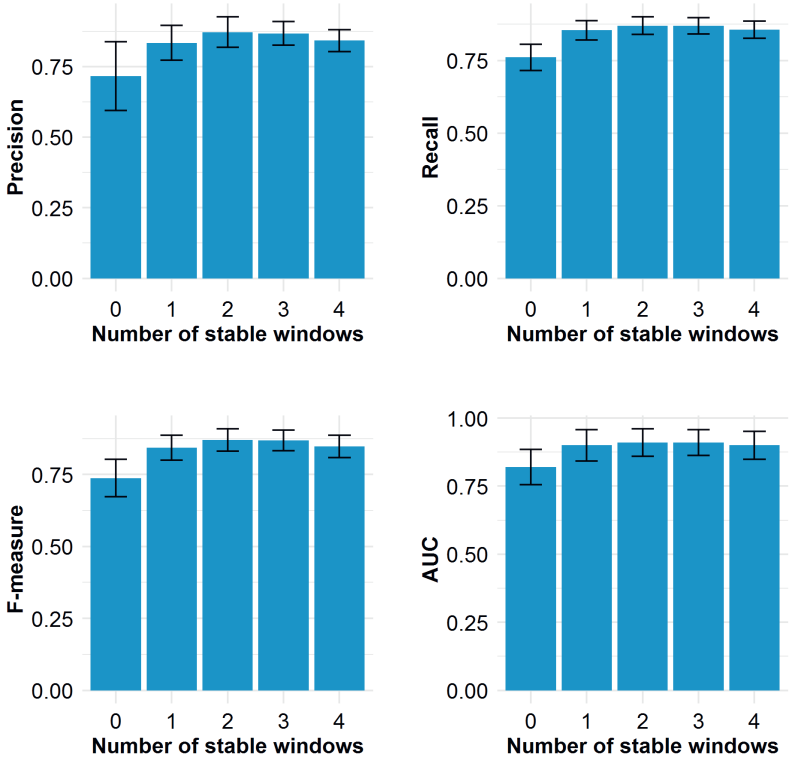


Figure 4.8: Evaluation results obtained for different levels of stability

The evaluation results of our predictive models show that the use of team features leads to a significant improvement in the predictions of delay. Moreover, our results show that team features can help improve the prediction of delay likelihood, while delay duration can be explained exclusively using them. For delay likelihood, the feature sets are complementary to each other. This means that the probability of a delay occurring depends on a combination of technical and team-related aspects, while the impact of the occurred delay (i.e., how fast it is resolved) mainly depends on the characteristics of the team. We therefore recommend organizations to encourage and facilitate teams to improve their work allocation, effort estimates, knowledge sharing and accountability in order to reduce the impact of delays. Companies must also have a stable ecosystem in place to ensure that teams are able to operate effectively.

4.4.2 Implications for Researchers

Feedback mechanisms on team behaviors. Our predictive models can be used by teams as awareness or feedback mechanisms on their behavior patterns during planning. The support provided by our models can help teams to increase the awareness of their own behavioral habits (e.g., to reduce the bias towards over-optimistic estimates). This

is likely to foster productive behavior change and improve schedule estimates. To better realize the benefits of such mechanisms, there is a need for better tool support that can support agile teams in tracking their team behavior and improving the management of agile projects. Current agile project management tools lack advanced analytical methods that are capable of deriving actionable insights from project data for planning. An extension of existing tools with actionable information about team dynamics and the current existence of risks in a sprint would be beneficial. Initial work in this direction has been carried out by Kortum et al. [238].

Team dynamics monitoring. One of the key novelties in our approach is deriving new team features for a user story by aggregating the features either at team- or individual-level. We derived the features by using a range of statistics over the past stories delivered by a specific team or developer. Our experimental results demonstrate the effectiveness of this approach. Previous research [239, 240] has shown that team-related information is often difficult to capture or not available due to lacking information sources. As a consequence, these factors are often not monitored. Our results indicate the significant benefits of incorporating such data into analysis/predictive models for effort estimation and planning in agile projects. Further research on team monitoring approaches is needed to address this gap and to gain a better understanding of what information and metrics can be collected across software organizations.

Impact of social-driven factors. The set of team-related factors identified in this paper are by no means comprehensive to encompass all aspects of teamwork. We were limited to the repository data available at ING. It is an interesting opportunity for future work to analyze the effects of social-driven factors related to the collaborative nature of software development work. Previous studies [154, 213, 224] have reported on the impact of social-driven factors, such as trust, team leadership, team cohesion and communication. These factors would be a good starting point for future work.

Applicability of effort prediction models. Our evaluation results show that our predictive models perform significantly better for teams that have been stable for a longer period of time. In our analyses, we did not make a distinction between someone leaving or joining the team, nor did we take into account the number of people leaving or joining. It is an interesting opportunity for future work to analyze the effects of different types of team changes on the models' performance. This could also include external changes, such as organizational restructuring or changes in senior management. Such changes have been identified as risk factors in literature [90–92]. Future research should also explore the impact of other team characteristics (e.g., team experience and seniority) on the performance of effort prediction models.

4.5 Threats to Validity

Construct validity. We consider data variables as constructs to meaningfully measure delay and risk factors. This introduces possible threats to construct validity [176]. We mitigated these threats by collecting real-world data from user stories and teams, and all

the relevant historical information available. The ground-truth (i.e., the delay in terms of the number of sprints overrun) is based on the number of sprints a story has been part of. However, it might happen that teams close their stories too early or too late. We cannot account for the impact of poor record keeping on our results. Even though all teams at ING are encouraged to deliver on-time, there is a possibility that some teams treat their delivery deadlines less seriously than others. These teams might add stories to sprints without the commitment to deliver on-time.

Internal validity. Our dataset has the class imbalance problem. This has implications to a classifier's ability to learn to identify delayed stories. To overcome this issue, we used the weighted variants of performance measures and employed AUC which is insensitive to class imbalance. We applied statistical tests to verify our assumptions [191], and followed best practices in evaluating and comparing predictive models for effort estimation [39, 53, 241]. However, we acknowledge that more advanced techniques could also be used, such as statistical over-sampling [242]. Another threat to our study is that the patterns in the training data may not reflect the situation in the test data. To mitigate this threat we used time-based cross-validation to mimic a real prediction scenario.

External validity. As with any single-case empirical study, external threats are concerned with our ability to generalize our results. We have considered 765,200 user stories from 571 teams, which differ significantly in size, composition, products developed and application domain. Although we control for variations using a large number of projects and teams, we acknowledge that our data may not be representative of software projects in other organizations and open source settings. Agile teams in other settings might have a different team structure and may work at a different pace or create stories differently. Further research is required to confirm our findings in different development settings.

4.6 Related Work

Teamwork in agile development. Previous research has analyzed the nature of agile teams in software development: their characteristics, how they collaborate, and the challenges they face in geographically and culturally diverse environments [243, 244]. A survey of success factors of agile projects identified team capability as a critical factor [9]. Other studies [104, 106] have used performance models to investigate the impact of teamwork quality on project success. Moe et al. [104] showed that problems with team orientation, coordination, work division and conflict between team and individual autonomy are important barriers for achieving team effectiveness. Melo et al. [220] found that team structure, work allocation and member turnover are the most influential factors in achieving productivity in agile projects. Our study complements prior work by exploring the effects of various aspects of teamwork in the context of predicting software delays.

Effort estimation and planning. A great body of research has been published on the study of effort estimation methods [6]. Estimation methods that rely on expert's subjective judgement are most commonly used in agile projects [12]. Project factors and personnel factors are the top mentioned effort drivers in agile projects [10, 12].

Recent efforts have leveraged machine learning techniques to support effort estimation and planning in software projects. They have achieved promising results in estimating effort involved in resolving issues [18–20], predicting the elapsed time for bug-fixing or resolving an issue (e.g., [23, 66, 67, 169]). Previous research [21, 22, 101] has also been done in predicting the delay risk of resolving an issue in traditional development. Some studies have been dedicated to effort estimation for agile development at the level of issues [18–20] and iterations [24, 25]. Our work specifically focuses on predicting delays in user stories, and complements prior work by introducing team features and incremental learning to follow team changes over time.

4.7 Conclusion

Modern agile development settings require different approaches to planning due to their iterative and team-oriented nature. In this paper, we explored the effects of various aspects of teamwork on delays in software deliveries. We extracted a set of technical and team-related risk factors that characterize delayed user stories. Based on these factors, we built models that can effectively predict the likelihood and duration of delays in user stories. The evaluation results demonstrate that:

1. The use of team features leads to a significant improvement in the predictions of delay likelihood, while delay duration can be predicted exclusively using them.
2. Developer workload, team experience, team stability and past effort estimates are the most important predictors for delays in user stories.
3. Training on recent user stories leads to more accurate and robust predictions of delay.
4. Our predictive models perform significantly better and more consistently for teams that have been stable.

Overall, our results indicate that planning in agile software development can be significantly improved by incorporating team-related information and incremental learning methods into analysis/predictive models. We identified several promising research directions related to team dynamics monitoring, designing feedback and awareness mechanisms on team behaviors, and the applicability of effort prediction models in agile projects. Progress in these areas is crucial in order to better realize the benefits of agile development methods.

5

Context-Aware Automated Sprint Plan Generation

Sprint planning is essential for the successful execution of agile software projects. While various prioritization criteria influence the selection of user stories for sprint planning, their relative importance remains largely unexplored, especially across different project contexts. In this paper, we investigate how prioritization criteria vary across project settings and propose a model for generating sprint plans that are tailored to the context of individual teams. Through a survey conducted at ING, we identify urgency, sprint goal alignment, and business value as the top prioritization criteria, influenced by project factors such as resource availability and client type. These results highlight the need for contextual support in sprint planning. To address this need, we develop an optimization model that generates sprint plans aligned with the specific goals and performance of a team. By integrating teams' planning objectives and sprint history, the model adapts to unique team contexts, estimating prioritization criteria and identifying patterns in planning behavior. We apply our approach to real-world data from 4,841 sprints at ING, demonstrating significant improvements in team alignment and sprint plan effectiveness. Our model improves team performance by generating plans that deliver more business value, align more closely with sprint goals, and better mitigate delay risks. Overall, our results show that the efficiency and outcomes of sprint planning practices can be significantly improved through the use of context-aware optimization methods.

This chapter has been published as *E. Kula, A. van Deursen, and G. Gousios. Context-Aware Automated Sprint Plan Generation for Agile Software Development, IEEE/ACM ASE'24 [132].*

Effective planning is crucial for the successful execution of software development projects [9]. Central to the planning is the ability to prioritize and select software features that deliver the most value to customers while mitigating delays. Over the past two decades, agile methodologies have become increasingly popular for managing software projects [11, 245]. Agile uses an iterative approach, enabling teams to manage changing priorities, rapidly deliver business value, and inherently reduce risks. However, effective planning remains challenging in agile settings. Previous research indicates that nearly half of agile projects exceed their timelines by 25% [12] and deliver 56% less business value than anticipated [4], highlighting the need for improved planning strategies.

In agile settings, software is developed incrementally through short iterations known as *sprints* [38]. Each sprint involves completing a subset of requirements, expressed as *user stories* [14]. Before a sprint begins, the team performs *sprint planning* to define the sprint goal and select user stories from the backlog. Various factors, referred to as *prioritization criteria*, such as business value and urgency, are used to prioritize and select user stories for sprint planning [109, 110, 246, 247]. Although business value is typically considered the main prioritization criterion in agile methods, previous research [107, 108] suggests that this may not always reflect actual practice. Sprint plans are developed by teams based on their cumulative knowledge and biases, making them specific to the context of each team. The relative impact of the prioritization criteria remains largely unexplored, especially across different project contexts.

The sprint planning process is complex and time-consuming, particularly for large projects where backlogs can grow to hundreds of user stories [111]. Agile teams would benefit from automated support that estimates prioritization criteria and generates sprint plans tailored to their specific context. Existing models [248–250] generate sprint plans based on team estimates of prioritization criteria, aiming to maximize business value. Some studies [251, 252] have extended these models to consider additional objectives, such as maximizing sprint goal alignment and capacity usage. However, existing approaches rely on team estimates of prioritization criteria and do not account for contextual influences. Recent studies (e.g., [18, 131]) suggest that machine learning techniques can improve software project management by providing contextual support and insights from project data. This has the potential to enhance the efficiency and outcomes of sprint planning.

The goal of this paper is to develop a model for generating sprint plans that align with the specific goals and performance of individual teams. To achieve this, we conducted a case study at ING, following the study design outlined in Figure 5.1. We start by investigating how prioritization criteria affect the selection of user stories for sprint planning. We conduct a survey with 52 teams to assess how they weigh the importance of these criteria and how this is influenced by project characteristics. Next, we collect historical backlog data from 4,841 sprints and use machine learning techniques to estimate prioritization criteria. We then develop an optimization model that integrates teams' planning objectives and sprint history to generate sprint plans tailored to each team's specific context. The model learns from past team performance to identify and incorporate planning behavior patterns. We evaluate our model through both quantitative and qualitative analyses. For the quantitative analysis, we use the historical backlog data to assess the model's effectiveness and alignment with team planning. We compare the performance of our model to the state-of-the-art in automated sprint planning methods. For the qualitative analysis,

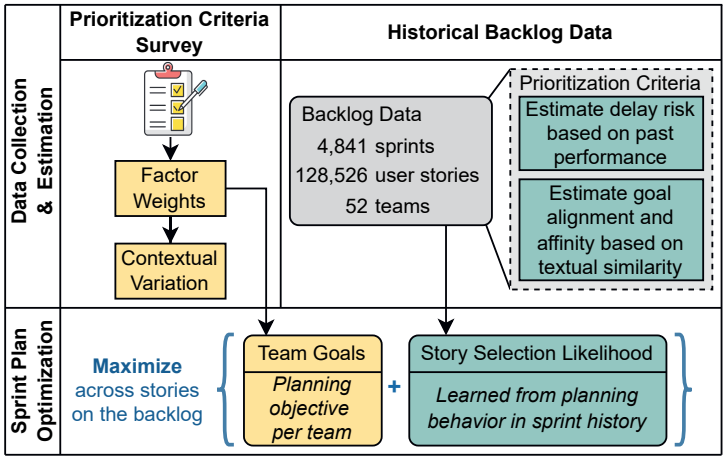


Figure 5.1: Overall study design: We collect survey data (shown in yellow) to obtain teams' weightings of prioritization criteria and their planning objectives. We collect historical backlog data (shown in gray) and use machine learning techniques (visualized in green) to estimate prioritization criteria. We develop an optimization model, integrating teams' planning objectives and behavior, to generate sprint plans tailored to each team's specific context.

we interview teams to gather insights into their perceptions of the model's usability.

Our survey results show that urgency, sprint goal alignment, and business value are the most important prioritization criteria, with their influence depending on project characteristics, such as project resources, priority, client type, and security level. The quantitative evaluation of our model demonstrates significant improvements in team alignment and sprint plan effectiveness. On average, the model achieves an 88% overlap in selected stories with the team's actual sprint plans, and a 74% semantic relatedness between differing stories. Our model outperforms the state-of-the-art and improves team performance by generating sprint plans that deliver 29% more business value, exhibit 14% stronger alignment with sprint goals, and reduce delay risk by 42%. In the qualitative evaluation, the majority of teams found our approach to be consistent with their goals and valuable as interactive support.

The main contributions of this paper are:

- A set of prioritization criteria ordered by their importance for sprint planning (Section 5.2.3).
- A context-aware optimization approach to generate sprint plans that align with team goals and performance (Section 5.3).
- An empirical evaluation of the approach and comparison to the state-of-the-art, demonstrating significant improvements in team alignment and sprint plan effectiveness (Section 5.4.2–5.4.4).
- A qualitative analysis of the approach with software teams identifying areas for future research (Section 5.4.5).

5.1 Background and Related Work

5.1.1 Sprint Planning

In agile software projects, *sprints* typically span 2–4 weeks [38]. During this period, teams design, implement, test, and deliver a product increment, such as a working milestone. Each sprint requires the completion of a set of *user stories*, which are brief descriptions of features written from the perspective of the end user. Teams maintain a prioritized list of pending user stories, known as the *product backlog* [35]. The product owner organizes the backlog by sorting user stories according to their urgency, ensuring that the most urgent stories are prioritized at the top. The urgency of a story is determined based on immediate customer needs and project deadlines. Each user story includes a title, a textual description clarifying the task, and several standard fields detailing the story's type, business value, and dependencies.

Before each sprint, teams hold a *sprint planning* meeting, divided into two parts. In the first part, the product owner and development team establish the sprint goal and discuss user stories in the backlog. The sprint goal is a concise statement that describes the primary focus of the sprint, such as implementing a new feature. In the second part of the meeting, the team breaks down user stories into specific tasks and estimates the effort required for each user story. Teams rely on expert judgement [10] to estimate effort, often using *story points* as the unit of measure. Story points reflect the relative effort, complexity, and risks associated with a user story [13]. The team then selects user stories to implement in the upcoming sprint based on prioritization criteria (described in Section 5.2.1). These criteria consider the potential value of the user stories and the risks associated with their execution. Additionally, the selection process accounts for development constraints, such as dependencies and the team's delivery capacity. Teams measure their capacity for future sprints by monitoring their *velocity*, which represents the average number of story points completed in previous sprints [13].

5.1.2 Related Work

Research on automated sprint planning (e.g., [248–250]) treats the process as an optimization problem. Previous studies have used methods such as mathematical programming [248–250] and genetic algorithms [251, 252] to select the optimal set of user stories for a sprint. These approaches aim to maximize business value over the selection of user stories, relying on team estimates of prioritization criteria and using a weighted sum of these criteria as the objective function. Golfarelli et al. [248] developed an extensive objective function that incorporates criticality risk and affinity as factors influencing the business value of user stories. Their work represents the state-of-the-art benchmark, which we use for our model evaluation in Section 5.4. Al-Zubaidi et al. [251] and Ozcelikkan et al. [252] extended their focus to multi-objective optimization, incorporating additional objectives such as maximizing sprint goal alignment and capacity usage. While most efforts address planning for individual sprints, recent studies (e.g., [246, 252]) have developed models for multi-sprint plans that allow for re-planning during execution.

Despite these advancements, existing approaches have two main limitations: (1) they use generic objective functions that are not tailored to individual team contexts, and (2) they are not fully automated, as they rely on team estimates of prioritization criteria. Our

study addresses these gaps by providing automated, contextual support for generating sprint plans. Our model incorporates teams' planning objectives and sprint history to create plans aligned with team goals and performance. We use machine learning techniques to estimate prioritization criteria, thereby relieving teams from the task of manual estimation. To evaluate the performance of our model, we use extensive real-world data from our case company.

5.2 Prioritization Criteria Survey

We start by identifying prioritization criteria (i.e., factors that influence the prioritization and selection of user stories for sprint planning) through literature analysis and observations at the case company. We aim to address the following research questions:

- **RQ1.1 Factor weights:** How do teams weigh the importance of prioritization criteria for sprint planning?
- **RQ1.2 Weight variations:** How do project characteristics affect the weightings of prioritization criteria?

To answer these questions, we conduct a survey with 52 teams (Section 5.2.2), through which we assess the factor weights and how they vary across different project settings (Section 5.2.3).

5.2.1 Deriving Factors from Literature and ING

From our literature analysis and observations at the case company, we identified six prioritization criteria. These criteria specifically affect the *order* in which user stories are prioritized and selected for planning, excluding development constraints such as dependencies and team capacity. We derived five of these criteria from the literature, while "strategic alignment" emerged from discussions with teams at ING. Below, we explain these criteria in detail, with each factor name underlined:

- Business value refers to the potential impact of a user story on achieving business goals and satisfying customer needs [14, 222]. This factor is often considered the primary criterion in agile methods [158, 253]. However, this assumption has been debated in previous research [107, 108], suggesting that it may not always reflect actual practice and requires further research.
- Urgency measures the time sensitivity of a user story, determined by its relevance to critical customer needs or alignment with project deadlines [254, 255].
- Sprint goal alignment evaluates how well user stories contribute to achieving the overall objective of the sprint [251, 256].
- Affinity measures the degree of similarity or relatedness between user stories within a project [246, 249]. High-affinity user stories share common themes or objectives, while low-affinity stories have less overlap in functionality or purpose. Delivering affine stories together in the same sprint can enhance the utility of the software functionality [248]. For example, a "data extraction" story may have limited value

on its own but becomes more valuable when delivered with a "data loading" story. Although affine stories complement each other, they are not interdependent and can be implemented separately.

- Delay risk indicates the likelihood of a user story experiencing delays, influenced by factors such as complexity, uncertainty, and impact on existing functionality [109, 110, 131]. User stories with high delay risk threaten deadlines, potentially leading to incomplete work and postponed feature delivery to customers.
- Based on discussions with teams at ING, we introduce strategic alignment as a new factor. This criterion evaluates how well user stories align with the organization's long-term strategic goals. During sprint planning meetings at ING, we observed that teams prioritized stories aligning with the company's strategic goals to improve product viability and promote software reuse.

5.2.2 Survey Setup

Survey Design

We developed an online survey to be completed collaboratively by software teams. The survey consisted of a mix of closed and open-ended questions; the final survey instrument is provided in the replication package [122]. To provide context, the survey's start page contained a brief outline of our study's purpose. The survey was divided into two main sections: the first section included multiple-choice questions to gather demographic data on the size, years of existence, geographic distribution, and application domains of the participating teams [136]. Additionally, teams were asked to provide their identification number from *ServiceNow*[1], the backlog management tool used at ING. This allowed us to link survey responses with project data to determine project settings.

In the second section, teams were asked to rank the prioritization criteria based on their importance for sprint planning in their current project. Teams were encouraged to discuss and determine the importance of the criteria collaboratively during a group meeting or at the start of a sprint planning session. They used a drag-and-drop format to rank the criteria, which were presented in random order to reduce ordering bias [138]. After ranking, teams were asked to collectively weigh the importance of each factor using a slider scale, with values ranging from 0 (not important) to 1 (highest importance). We designed the slider scale to be intuitive and easy for teams to adjust their weights. To ensure relative weighting, the total score across all prioritization criteria was limited to 1. At the end of this section, we included an open-ended question allowing respondents to suggest additional factors. We received nine responses, which we reviewed manually and found to be either rephrasings or sub-cases of existing prioritization criteria.

Survey Validation

We piloted the survey with five randomly selected teams from ING TECH to refine the questions [135]. The pilot version featured an additional open-ended question for feedback on the survey content. All five teams provided feedback, highlighting the need for slider scales to assign factor weights and revealing ambiguity in the factor names. Based on their input, we provided definitions for the prioritization criteria in the final survey.

[1]https://www.servicenow.com/

Survey Execution
Our target population consisted of all 301 software teams within ING TECH. We accessed a mailing list containing 115 product owners representing these teams. For the final survey, we excluded the five teams and their respective product owners involved in the pilot run. In June 2023, we distributed the survey to the remaining 112 product owners and their 296 teams. We sent personal invitation emails to the product owners, explaining the survey's purpose and requesting them to complete the survey with their teams. Participants had a total of three weeks to respond. We received responses from 52 teams, resulting in a response rate of 17%. We sent reminders halfway through the second week to follow up on non-responders.

Survey Demographics
The survey gathered demographic information about the teams. The majority of teams (92%) reported to consist of five to eight members. Team existence ranged from one year (18%) to over five years (21%), with a median of three years. Additionally, 86% of teams indicated having had the same product owner over the past year, and 32% reported being globally distributed, conducting sprint planning meetings online. The teams worked in diverse application domains: web (27%), mobile (18%), desktop (11%), cloud-based (21%), and AI/data science (23%).

Survey Data Analysis
To investigate how teams perceive the importance of prioritization criteria (RQ1.1), we visualize the distributions of rank order responses and use descriptive statistics to compare factor weights. To assess whether factor weights are affected by project characteristics (RQ1.2), we investigate the impact of two main attributes: project size and project type. Previous research suggests that these attributes can affect requirements engineering approaches in agile projects [108, 257]. For project size, we measure resource availability in terms of the number of people assigned to the project, the time duration, and the budget allocated. For project type, we assess the priority assigned to the project, whether the client is internal or external to the case company, and whether the project requires resource-intensive security testing. These characteristics are used at ING to classify projects by size and type for planning and resource allocation. An overview of the extracted project characteristics and their descriptions is provided in Table 5.2. Using team ID numbers from the survey responses, we link the respondents' factor weightings to their corresponding projects in the backlog management data. We extract the project characteristics directly from the primitive attributes of the project in the data. We then conduct a correlation analysis to determine how these project characteristics affect the assigned factor weights. Since our data is not normally distributed, we use Spearman's rank correlation [258] and apply Holm's correction [218] to adjust for multiple comparisons.

5.2.3 Survey Results

(RQ1.1) Factor Weights
Table 5.1 presents the distributions of rank order responses and factor weights assigned by respondents. The "Rank" column indicates the order of factors by their weighted average rank scores, ranging from rank 1 (most important) to rank 6 (least important). Urgency

ranked first, sprint goal alignment second, and business value third, with weighted average rank scores of 2.72 or lower. Over 67% of teams ranked urgency and sprint goal alignment as the most or second most important factors, with median weight scores of 0.26 or higher. Less than half (48%) of the teams ranked business value as one of the top two factors, yet it received a high median weight score of 0.21. Affinity and delay risk were perceived as being less important, with weighted average rank scores of 4.33 or higher, and notably lower median weight scores of 0.12 or less. Strategic alignment was ranked as the least or second least important factor by 82% of teams. Further analysis shows consistent rankings for sprint goal alignment and strategic alignment, with standard deviations lower than 1.05. There was greater variability in the rankings for other factors, with standard deviations ranging from 1.15 to 1.23.

> Urgency, sprint goal alignment, and business value are the top most important prioritization criteria. Each is perceived to contribute more than 20% to determining the priority of a story.

5

Table 5.1: Overview of the prioritization criteria and their perceived importance for sprint planning. Teams ranked the factors by importance and assigned weights using values from 0 (not important) to 1 (highest importance). *Rank distribution* shows the distributions of rank order responses, with rank 1 being the most important rank and rank 6 the least important. *WA* represents the weighted average of factor ranks. *Median weight* and *95% CI* indicate the median and 95% confidence interval of the weights assigned to the factors. The overall *Rank* is determined by the order of the weighted averages.

Factor	Rank distribution 1 2 3 4 5 6	WA	Median weight	95% CI	Rank
Urgency		1.92	0.33	[0.18, 0.45]	#1
Sprint goal alignment		2.09	0.26	[0.14, 0.41]	#2
Business value		2.72	0.21	[0.11, 0.38]	#3
Affinity		4.33	0.12	[0.04, 0.19]	#4
Delay risk		4.51	0.10	[0.04, 0.16]	#5
Strategic alignment		5.25	0.07	[0.02, 0.11]	#6

Table 5.2: Results of the correlation analysis between project characteristics and the weights assigned to the prioritization criteria in survey responses. The prioritization criteria are abbreviated as follows: urgency (UR), sprint goal alignment (SG), business value (BV), affinity (AF), delay risk (DR), and strategic alignment (SR). Spearman's correlation [258] coefficients are used to show relationship strengths, depicted by colors indicating weak, moderate or strong relationships. Statistical significance is indicated with * at the 0.05 level after Holm correction [218].

Project attribute	**Project characteristic**	**Description of how we measured the characteristic at ING**	**Type**	**Spearman's correlation coefficients**					
				UR	SG	BV	AF	DR	SA
Size	People	Total number of people working on the project	Continuous	0.41*	0.53*	0.46*	−0.11	−0.18	0.29*
	Time	Planned project duration in days	Continuous	−0.30*	0.27*	0.38*	−0.14	−0.49*	0.41*
	Budget	Total estimated monetary project costs	Continuous	0.25	0.52*	0.58*	0.16	−0.32*	0.36*
Type	Priority	Assigned priority class: 1. low prio, 2. moderate prio, 3. high prio	Categorical	0.58*	0.43*	0.49*	0.18*	0.54*	−0.25*
	Client type	Whether the project's client is external to ING	Binary	0.45*	0.37	0.67*	0.42*	0.51*	−0.39*
	Security level	Whether the project requires resource-intensive security testing	Binary	−0.54*	−0.44*	−0.42*	0.61*	0.45*	−0.38*

(RQ1.2) Weight Variations
Table 5.2 presents the results of the correlation analysis between project characteristics and the weights assigned to the prioritization criteria by survey respondents. Significant correlations indicate variations in factor importance across different project settings. Teams working on projects with abundant resources assign significantly higher weights to sprint goal alignment, business value, and strategic alignment. They show less concern for affinity and delay risk, likely because they have sufficient resources to mitigate the consequences of fragmented and late deliveries. Teams working on high-priority projects, with external clients, or on security-critical systems prioritize delay risk and affinity more, while assigning less weight to strategic alignment. Specifically, high-priority projects and external client projects prioritize the customer-focused criteria, urgency and business value, whereas security-critical projects place more emphasis on affinity, possibly for end-to-end testing.

> The importance of prioritization criteria varies significantly based on project characteristics, such as resources, priority, client type, and security level. *This variation demonstrates the need for contextual support in sprint planning practices.*

5.3 Modeling Story Prioritization and Sprint Plan Optimization

The variations in factor weights across teams highlight the need for contextual support in sprint planning. To address this need, we aim to develop a model that generates sprint plans tailored to the specific context of each team. We use survey responses and sprint history data to model each team's planning objectives and past performance. First, we collect historical backlog data (Section 5.3.1) and apply machine learning techniques to estimate the prioritization criteria for user stories (Section 5.3.2). To capture planning behavior, we develop a machine learning model that learns from a team's sprint history to predict the likelihood of the team selecting a particular story for their upcoming sprint (Section 5.3.3). We derive team planning objectives from the prioritization criteria and their corresponding weights provided in the survey responses (Section 5.3.4). Combining these elements, we build an optimization model based on linear programming (Section 5.3.5). As illustrated in Figure 5.1, the optimization is guided by an objective function composed of two components: the team's planning objective, which reflects their goals, and a selection likelihood estimate, which reflects their planning behavior. By optimizing this objective while adhering to development constraints, our model selects the optimal set of user stories tailored to the team's context for the upcoming sprint.

5.3.1 Backlog Data Collection

Backlog data
To develop and evaluate our model, we require a dataset containing historical records of each team's backlog and sprints. For each sprint, this dataset should include the *identification number*, *start date*, *end date*, *team velocity*, the textual *sprint goal* field, and the set

of user stories selected for that sprint. Similarly, user stories should include their *identification number*, *urgency*, *business value*, *story points*, *dependencies*, and the textual *title* and *description* fields. Since story contents might change before a sprint begins, we capture the information recorded on the day of sprint planning. This ensures consistency with the data available to the team during planning. For each team, we extract snapshots of their backlog on the days of sprint planning, linking team ID numbers with user stories to obtain the list of stories available. If a story is associated with a sprint ID number, it indicates that the story has been planned; if not, it remains unplanned in the backlog.

At ING, we extracted log data from the backlog management tool *ServiceNow*. This dataset contains records from 4,841 sprints and 128,526 user stories from the 52 respondent teams, covering the period from January 1, 2019 to January 1, 2023.

Data pre-processing

We took several steps to eliminate noise and address missing values in the data. First, we filtered out sprints with a status other than 'Completed', focusing on fully executed sprints. We then removed sprints that underwent significant content alterations during development, as these instances are likely unstable. After cleaning the data, the final dataset reduced to 4,812 sprints from 52 teams.

5.3.2 Estimating Prioritization Criteria

To model story prioritization, we need to measure or estimate the prioritization criteria for past user stories. Our approach is as follows:

Business value: We extract business value directly from the primitive attributes of the stories in the dataset. This value is represented by a numerical score between 1 and 10, assigned by the product owner, indicating its perceived value to the customer. A higher score denotes greater business value.

Urgency: Urgency is derived from the position of the story in the backlog, with higher positions indicating greater urgency.

Sprint goal alignment: To measure sprint goal alignment, we assess the textual similarity between the content of each user story and sprint goal statement. For each user story, we concatenate the title and description into a single text document. For the sprint goal statement, we use the text as a separate document. We then pre-process these documents by converting the text to lowercase and removing punctuation and stop words. Using the Doc2Vec technique [230], we generate fixed-length vector representations for both the user stories and the sprint goal documents. To quantify the alignment, we calculate the cosine similarity between the embeddings of the user stories and the sprint goals.

Affinity: To measure the relatedness between user stories, we use a procedure similar to the one used for assessing sprint goal alignment. Instead of calculating the similarity between the embeddings of user stories and sprint goals, we compute the cosine similarity between the Doc2Vec-generated embeddings of the user stories themselves. The resulting matrix of cosine similarity scores provides insights into the affinity among the stories in the backlog. To calculate the affinity score for a sprint, we sum the cosine similarity scores for all unique pairs of user stories selected for the sprint and normalize this sum by the

number of pairs. For each pair of stories on the backlog, we multiply their similarity score by the product of their selection variables.

Delay risk: To estimate delay risk, we follow a procedure outlined by Kula et al. [131] for predicting delay likelihood in user stories. Their method achieved an average F1 score of 76–84% across a large industrial dataset. We extract the 13 most significant predictor variables, identified as having an importance value higher than 0.05 in the study of Kula et al., and augment these with the Doc2Vec-generated story embeddings as an additional input feature. Since Kula et al. utilized the same backlog management tool, we were able to directly extract these variables from ING's data using the same methodology. A detailed overview of the extracted variables is provided in the replication package [122]. We focus exclusively on user stories marked as 'Completed' in the backlog data, classifying a story as 'delayed' if it was postponed for one or more sprints. To simulate a realistic planning scenario, we extract predictor variables as they were recorded on the day of sprint planning for the sprint to which the story was originally assigned.

For model building, we compare and evaluate four different classifiers that have been shown to be effective in risk prediction: Random Forests [232], AdaBoost [233], Multilayer Perceptron [234] and Naive Bayes [22, 235]. A summary of the evaluation results can be found in the replication package [122]. A comparison shows that Random Forests outperforms the other classifiers. Therefore, we employ Random Forests and build predictive models tailored to each team, trained and tested on individual teams' backlogs. We sort the user stories chronologically by their start dates, and use a 70-30 split for training and evaluation. The initial 70% of the stories are allocated to the training set, and the remaining 30% to the test set. This approach ensures that the model learns from historical data preceding the stories it is tested on.

Strategic alignment: We were unable to measure or find proxies for strategic alignment. ING does not collect quantitative data on this factor nor has a standardized method for strategy reporting.

5.3.3 Predicting Story Selection Likelihood

We develop a method to learn story selection based on team planning behavior. Specifically, we build models that learn from a team's sprint history to predict the likelihood of the team selecting a particular story for the upcoming sprint. These models identify the types of stories, or combinations of prioritization criteria, that teams select for their sprints under given constraints. We use *binary classification* and build team-specific models, training and testing them with historical backlog data from a specific team. For each sprint, we extract a historical snapshot of the backlog as recorded on the day of sprint planning, with the corresponding stories serving as input instances for the model. The number of user stories used for training each team-specific model ranges from 1,287 to 2,050. Input features include the *team velocity* set for the sprint, the estimated prioritization criteria for the stories, and the number of outgoing *dependencies* for each story on the backlog. Stories are labeled based on whether they were selected for the respective sprint.

We evaluated four machine learning algorithms suitable for classification tasks in software project management: Random Forests [18, 131], Multi-Layer Perceptron [259, 260],

Least Median Square [259], and Naive Bayes [22]. A comparison of their predictive performance demonstrated that Random Forests outperforms the other classifiers, with an average improvement of 6–18% in precision, 10–24% in recall, and 7–20% in F1 score. Therefore, we chose Random Forests for our experimental setup.

To simulate a real planning scenario, where decisions rely on insights from previous sprints, we sort the sprints chronologically by their start dates. For training and evaluation, we use a 70-30 split: the initial 70% of sprints are allocated to the training set, and the remaining 30% are used for the test set.

5.3.4 Obtaining Team Planning Objectives

We derive team planning objectives from the weights assigned to the prioritization criteria in the survey. To account for the absence of strategic alignment, we re-scale the weights of the remaining factors. We then convert the factors and their adjusted weights into a weighted sum, which represents the team's planning objective. This objective reflects the factors in the proportion that teams aim to optimize when selecting user stories for the upcoming sprint, with higher weights indicating greater importance. For example, team $t48$ provided the following weightings: 0.30 for urgency, 0.20 for sprint goal alignment, 0.20 for business value, 0.15 for delay risk, 0.10 for affinity, and 0.05 for strategic alignment. We re-scale and convert these weights into the following planning objective, which is to be optimized over a backlog of N user stories:

$$\begin{aligned}\text{Objective t48} = \max \sum_{i=1}^{N} & 0.31 \cdot \text{urgency}_i + 0.21 \cdot \text{business value}_i \\ & + 0.21 \cdot \text{sprint goal alignment}_i + 0.11 \cdot \text{affinity}_i \\ & - 0.16 \cdot \text{delay risk}_i\end{aligned}$$

All factors, except delay risk, contribute positively to the value of user stories and should be maximized in the selection process for sprint planning. In contrast, delay risk should be minimized, which is why it is assigned a negative coefficient.

5.3.5 Optimization Model Development

We formalize sprint plan optimization using the 0-1 knapsack problem, where sprints are treated as knapsacks and user stories as items. The capacity of each sprint is defined by the team's velocity, and each story's weight is measured in story points. The objective is to select a subset of user stories that maximizes both the team's planning objective and the estimated story selection likelihood. This subset must adhere to constraints related to the team's velocity and story dependencies. We propose a linear programming model [261, 262] with the following variables:

- $u_i = 1$ if user story i is selected for the upcoming sprint, $u_i = 0$ otherwise;
- $likelihood(i)$ is the probability that story i will be selected by the team for the upcoming sprint, as predicted by our model using historical sprint data;

- $velocity$ is the team's capacity, i.e. the number of story points a team can deliver in a sprint;
- $story\ points_i$ is the number of story points assigned to story i;
- D_i is the set of user stories that story i depends on and that have not been completed yet;

The goal is to maximize the following objective function z by optimizing the assignment of the u_i variables over a backlog of N stories:

$$\text{z} = \max \sum_{i=1}^{N} \text{team planning objective} + \text{likelihood(i)} \tag{5.1}$$

subject to constraints:

$$\sum_{i=1}^{N} \text{story points}_i \cdot u_i \leq \text{velocity} \tag{5.2}$$

$$\sum_{j \in D_i} u_j \geq u_i \cdot |D_i| \tag{5.3}$$

The objective function z in Equation 5.1 aims to maximize the team planning objective and the selection likelihood estimate across the user stories on the backlog. Equation 5.2 constrains the total sum of story points for the selection of stories to not exceed the team's velocity. The ≥ symbol in Equation 5.3 ensures that all prerequisite stories for a given story are either completed beforehand or planned for the same sprint, thereby guaranteeing a logical sequence of story completion. We used the *Gurobi* solver in the Python library *PuLP*[2] to solve the linear programming problem.

5.4 Model Evaluation

Our evaluation aimed to answer the following research questions:

- **RQ2. Model alignment with team planning:** *Does the proposed approach generate sprint plans that align with teams' actual sprint plans?* To evaluate how well the sprint plans generated by our model align with team planning, we compare the overlap in selected stories between our model-generated sprint plans and those created by the teams against a state-of-the-art (SoTA) baseline. We also assess the impact of different components of our objective function by analyzing the model's performance using only the team planning objective, only the selection likelihood estimate, and the combined objective.
- **RQ3. Model effectiveness:** *How effective are the sprint plans generated by our model compared to teams' actual sprint plans in terms of value delivery and risk mitigation?* We evaluate the effectiveness of our model-generated sprint plans against those created by the teams. This assessment involves aggregating the prioritization criteria across the user stories within each sprint plan to measure their overall effectiveness.

[2]https://pypi.org/project/PuLP/

- **RQ4. Model usability:** *How do teams perceive the performance and usability of our model?* We apply the model to the current state of the backlogs and conduct interviews with teams to gather feedback and identify areas for improvement.

We address model alignment and effectiveness through a comparison with team's actual sprint plans and a SoTA baseline [246, 248], and model usability through a qualitative evaluation at ING.

5.4.1 SoTA Baseline

The single-sprint version of the linear programming model proposed by Golfarelli et al. [246, 248] serves as the state-of-the-art benchmark. The objective function of this model aims to maximize the cumulative business value of the user stories selected for the sprint. The business value of each story is increased if related (affine) stories are included in the same sprint or if the story is critical (i.e., urgent). Importantly, the objective function is fixed and does not account for team-specific contexts.

In their capacity constraint formulation, Golfarelli et al. increase the story points of user stories by their uncertainty risk. Similar to our dependency constraint, their model includes inequalities that ensure prerequisite stories for a given story are either completed beforehand or planned for the same sprint. Their model differentiates between AND-type and OR-type dependencies.

The single-sprint version of the linear programming model proposed by Golfarelli et al. defines the following variables:

Definition 1 (Sprint Planning Problem). Given a set of n user stories U in the backlog, let:

- $x_i = 1$ if and only if story i is included in the upcoming sprint, 0 otherwise;
- u_i be the utility (i.e., business value) of story i;
- p_i be the number of story points of story i;
- p^{max} be the capacity of the sprint (i.e., team velocity), measured in story points;
- r_i^{cr} be the criticality risk (i.e., urgency) of story i
- r_i^{un} be the uncertainty risk of story i;
- a_i be the affinity of story i;
- $Y_i \subset U$ be the set of stories related (affine) to story i;
- y_i be an auxiliary variable related to the number of stories in Y_i included in the upcoming sprint. y_i is zero if story i is not part of the sprint, otherwise it equals the number of stories in Y_i that are included in the sprint;
- $D_i \subset U$ be the set of stories on which story i depends and that have not been completed yet;

- $U \supseteq U^{AND} \cup U^{OR}$, where U^{AND} and U^{OR} are the subsets of stories having dependency type *AND* and *OR*, respectively;

The model aims to optimally assign the x_i variables. The linear programming formulation proposed by Golfarelli et al. is given by:

$$z = \text{Max} \sum_{i=1}^{n} u_i(r_i^{cr} x_i + a_i \frac{y_i}{|Y_i|}) \tag{5.4}$$

subject to constraints:

$$\sum_{i=1}^{n} p_i r_i^{un} x_i \leq p^{max} \tag{5.5}$$

$$\sum_{j \in D_i} x_j \geq x_i \qquad \forall i \in U^{OR} \tag{5.6}$$

$$\sum_{j \in D_i} x_j \geq x_i |D_i| \qquad \forall i \in U^{AND} \tag{5.7}$$

5

Adapting the Model to the ING Context

Golfarelli et al. use similar prioritization criteria but with different terminology. Specifically, their factors "utility" and "criticality" correspond to what we refer to as "business value" and "urgency." Consequently, in the context of the ING data, we measure utility and criticality as business value and urgency, respectively. In adapting the linear programming formulation to the ING context, we make two modifications:

- We use our delay risk factor as a proxy for the "uncertainty risk" factor defined by Golfarelli et al. They describe an uncertain story as one whose complexity is hard to estimate due to potential unexpected problems, a phenomenon that corresponds with our delay risk factor.
- At ING, all dependencies are of the AND-type. Therefore, we exclude Equation 5.6, which handles OR-type dependencies, as it is not applicable in the ING context.

The optimization model developed by Golfarelli et al. [246, 248] relies on team estimates for all prioritization criteria. However, obtaining retrospective team estimates for historical stories at ING was not feasible. Therefore, we use our procedure described in Section 5.3.2 to estimate the prioritization criteria.

5.4.2 Experimental Setup

We perform experiments using the test set that comprises 30% of the teams' past sprints (described in Section 5.3.3), incorporating our predictions of delay risk and selection likelihood for the user stories. To address RQ2, we compare the sprint plans generated by our model with those created by the teams in two ways. First, we assess the overlap of selected user stories using the *Jaccard Similarity* coefficient. This coefficient measures the proportion of user stories common to both our model's selection and the teams' selection,

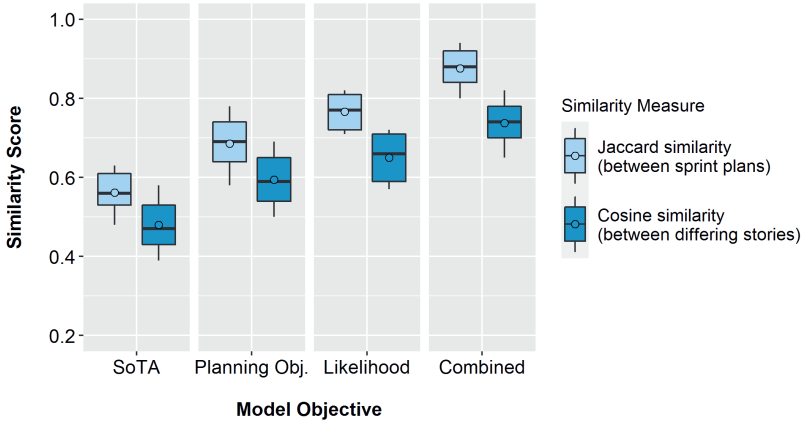


Figure 5.2: Comparison of story overlap (measured by Jaccard similarity; light blue) and semantic relatedness among differing stories (measured by cosine similarity; dark blue) between the teams' actual sprint plans and those generated by our model and the SoTA baseline. Results are shown for our model using the team planning objective (*planning obj.*) alone, the selection *likelihood* estimate alone, the *combined* approach, and the *SoTA* objective [246, 248].

relative to the total number of unique user stories across both sets. A Jaccard Similarity value close to 1 indicates high similarity, while a value closer to 0 indicates lower similarity or greater dissimilarity. Second, we assess the semantic relatedness among user stories that differ between the model's selection and the team's selection. We compute cosine similarity scores for the Doc2Vec-generated embeddings [230] of these differing stories to measure semantic relatedness.

For RQ3, we evaluate the overall effectiveness of sprint plans by aggregating the prioritization criteria across the user stories in each sprint plan. To ensure comparability of urgency scores across different product backlogs, we normalize the urgency ranks by the total number of stories in each backlog, thus obtaining a relative urgency score. For performance comparison, we use the Wilcoxon Signed Rank Test [191] to determine the significance of the evaluation results. We measure the effect size using Vargha and Delaney's $\hat{A}_{12}$ statistic [191], a non-parametric measure commonly used in software project management [62].

5.4.3 (RQ2) Model Alignment

Figure 5.2 presents the evaluation results, comparing the story overlap (Jaccard similarity scores) and semantic relatedness among differing stories (cosine similarity scores) between the teams' actual sprint plans and those generated by our model and the SoTA baseline. On average, our model, using the combined objective function, achieves a high Jaccard similarity of 0.88 and a cosine similarity of 0.74. It significantly improves the models using either one of the individual objective components, with improvements of 15%–27% in Jaccard similarity and improvements of 13%–24% in cosine similarity. Statistical tests

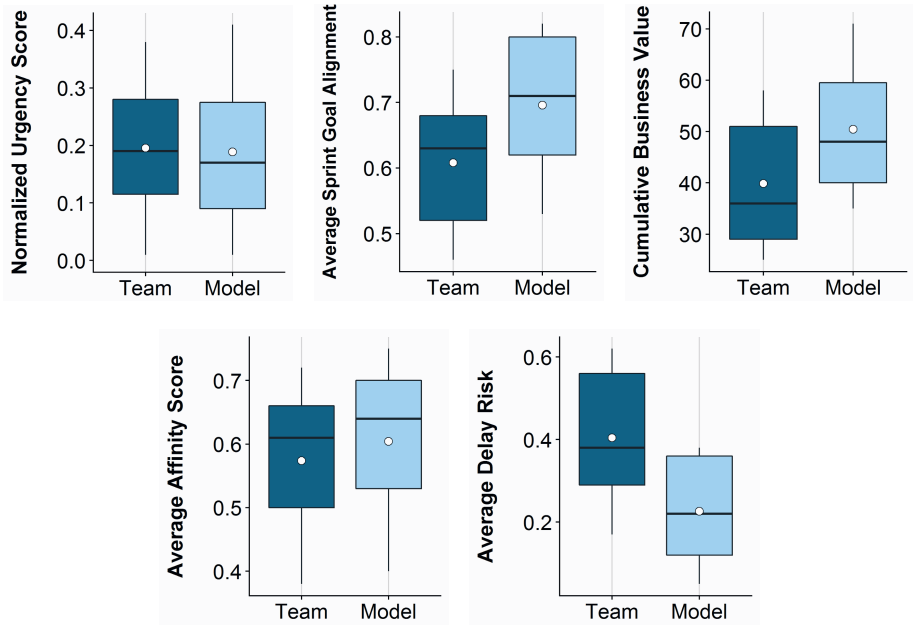


Figure 5.3: Comparison of aggregated prioritization criteria between *team*-composed and *model*-generated sprint plans. The *Normalized urgency score* reflects each story's backlog position divided by the total number of stories, with lower scores indicating greater urgency and more effective planning. *Cumulative business value* is the sum of business value scores assigned to stories in a sprint. *Average sprint goal alignment* and *Average affinity score* refer to the cosine similarity scores between the embeddings of user stories and sprint goals, averaged over stories in the sprint plans. *Average delay risk* represents the average probability of story delay, with lower values indicating more effective planning.

confirm significant improvements ($p < 0.001$) with medium to large effect sizes, ranging from 0.64 to 0.81. The model using the SoTA objective achieves the lowest scores, with an average Jaccard similarity of 0.56 and a cosine similarity of 0.48. All three versions of our model outperform the SoTA with large effect sizes greater than 0.81.

> The proposed approach, integrating teams' planning objectives and behavior, is effective in generating sprint plans that are aligned with teams' actual sprint plans.

5.4.4 (RQ3) Model Effectiveness

Figure 5.3 shows the distributions of aggregated prioritization criteria across sprint plans generated by our model and those created by teams. The model-generated plans demonstrate notable improvements in effectiveness compared to the teams' plans, with higher cu-

mulative business value, stronger sprint goal alignment, and more effective delay risk mitigation. On average, the model increases business value by 29%, improves sprint goal alignment by 14%, and reduces delay risk by 42%. Statistical tests confirm the significance of these improvements ($p < 0.001$), with medium to large effect sizes ranging from 0.66 to 0.87. Additionally, our model achieves an average improvement of 5% in affinity, which is significant ($p < 0.01$) but with a small effect size ($\hat{A}_{12} = 0.57$). The differences in normalized urgency rank scores are not significant.

> Our model drives improvements in team performance by generating sprint plans that deliver more business value, align more closely with sprint goals, and better mitigate delay risks.

5.4.5 (RQ4) Model Usability

Interview Methodology

The main goal of our qualitative analysis was to assess how teams perceive the performance and usability of our model. We conducted semi-structured interviews with 10 teams at ING to gather feedback and identify areas for improvement. We opted for a semi-structured format due to its flexibility in discussing prepared questions and exploring emergent topics [263]. Table 3 provides an overview of our interview questions. We primarily asked open-ended questions to encourage in-depth discussion without bias, as well as a focused question to examine the teams' willingness to use the model in practice. The study design was approved by the ethical review board of ING. We randomly selected and invited 10 teams from the pool of survey respondents to participate. We sent email invitations to the product owners of these teams, outlining the study's purpose and the required commitment. All teams accepted the invitation. Demographic details of the participating teams can be found in the replication package [122].

Table 3: Overview of Interview Questions

- **RQ4.1 Model alignment with team planning:** How well does the generated sprint plan align with your team goals and criteria for selecting stories?
- **RQ4.2 Model effectiveness:** Are there any modifications you would suggest for the generated sprint plan? If so, what changes would you propose and why?
- **RQ4.3 Model impact:** How do you foresee this model influencing your sprint planning?
- **RQ4.4 Model usability:** Would you use this model in practice? If so, how would you integrate it into your sprint planning process?

The interviews were conducted face-to-face by the first author at the start of the teams' sprint planning meetings. They lasted, on average, 41 minutes. Each interview began with a brief introduction to the study and a demonstration of the team's backlog. We applied our model to the current state of the backlog and presented the generated plan for the

upcoming sprint in a ServiceNow mock-up. This sprint plan served as a focal point for group discussion on model performance and usability.

We recorded and transcribed the interviews for analysis. We used *open coding* [116, 264] to analyze the transcripts and summarize the responses, coding by statement and allowing codes to emerge throughout the process. We constantly compared and refined the codes, grouping similar ones into categories.

Interview Results

This section presents the findings from our interviews. We use a [tX] notation to mark example quotes from the interviews, where 'X' refers to the identification number of the corresponding team. The codes resulting from our manual coding process are underlined.

RQ4.1: Model alignment with team planning. Consistent with our findings for RQ2, which showed a high Jaccard similarity score of 0.88, a majority of teams (seven out of ten) found that the model's story selections aligned with their reasoning and intuition. Teams described the model as making "*appropriate choices*" [t04] and "*intuitive selections*" [t09] that match their objectives and sprint priorities. For instance, one team stated: "*The proposed sprint plan primarily includes user stories that provide genuine value to the customer and that we would like to pick up in the next sprint.*" [t04] Another team noted that the model effectively captures a balanced value-risk trade-off: "*The model picked user stories that are valuable or urgent to the customer and ready for implementation. A few user stories at the top of our backlog still require further refinement before implementation can begin. The model seems to address that properly.*" [t08]

However, two teams had reservations regarding the implementation order of user stories. One team commented: "*While this [sprint] plan is very much in line with what our customers want right now, it contains several high-priority stories that we would normally divide across multiple sprints to dedicate sufficient attention to each.*" [t10] Another team perceived a lack of long-term perspective regarding the product's trajectory:"*The model does not differentiate between customer value and product value. It prioritizes customer needs but misses out on the bigger picture of where the product should be going.*" [t05] One team noted that their changing objectives, resulting from strategic shifts in their project, were not reflected in the generated sprint plan: "*Due to recent budget shifts, we have shifted our priority to smaller and low-risk user stories, differing from what we reported in the survey. The model's selections reflect our goals from the past year but do not align with our current situation.*" [t03]

RQ4.2: Model effectiveness. Consistent with our findings for RQ3, which showed increased effectiveness in the sprint plans generated by our model, most teams (six out of ten) indicated they would make only minor tweaks or refinements to the sprint plan rather than substantial alterations. Teams' suggestions regarding potential modifications revealed several themes. Four teams highlighted the need to postpone selected stories due to changes in team composition. For example, one team explained: "*Next sprint one of our senior members will be absent. While the model adapted by creating a smaller sprint, it selected two stories that rely on the expertise of the senior member. Our typical course of action is to postpone these stories until the required developer returns.*" [t05] Three teams preferred to postpone selected stories that require further refinement and are not yet ready

for implementation. For instance, one team mentioned: "*The proposed plan includes three stories for which we currently lack a clear approach. We still need to do some research and break these stories down into smaller tasks to gain a better understanding of the required solution.*" [t01] Other mentioned modifications included adjusting the order of user stories, such as advancing risky stories to early sprints to avoid late side-effects.

RQ4.3: Model impact. Teams identified three key benefits of integrating our model into their sprint planning process. Firstly, they noted that the model could improve productivity by facilitating quicker decision-making. For example, one team stated: "*The model guides teams to focus their discussions on the inclusion of pre-selected stories, reducing the need for lengthy, exploratory conversations typical of creating a sprint from scratch.*" [t02] Secondly, teams believed that the model could enhance business alignment in sprint planning. A team member explained: "*The model could help us focus on customer needs rather than internal interests.*" [t06] Lastly, teams suggested that the model could facilitate informed decision-making by "*providing insights into the backlog*" [t01] and stimulating discussions on overlooked stories. One team stated: "*We discussed stories that had been lingering in the backlog for a while and were typically skipped during our meetings.*" [t04] However, some teams also mentioned potential drawbacks of the model, such as overreliance on the model and reduced communication among team members. One team explained: "*Over time, teams might become dependent on the model, resulting in decreased communication within the team and limited exploration of alternative approaches.*" [t06]

RQ4.4: Model usability. Eight teams expressed their willingness to use the model as part of their sprint planning meetings. They see the model as a support for human judgement, to be used in conjunction with existing planning strategies, rather than as a replacement. The teams emphasized the importance of maintaining control over the sprint plan and having the flexibility to adjust it as needed. They favored an interactive format that allows them to modify the sprint and team objectives on the go. Some teams stressed the importance of model explainability, suggesting that the model would be more useful if it provided explicit rationale for story selection. For instance, one team explained: "*We would like to understand why the model selects this particular set of user stories and how it affects the feasibility of the sprint if we decide to choose other stories.*" [t07] The two teams that were hesitant to use the model in practice expressed concerns about overreliance on it, fearing it could diminish the quality of team discussions.

5.5 Discussion

5.5.1 Recommendations for Practitioners

Improving sprint planning. Our model improves the efficiency and outcomes of sprint planning. We identified a set of key factors and project context variables that influence story prioritization. By incorporating these factors into an optimization model, teams can generate context-aware sprint plans that align with their goals and performance. Using a combination of team-supplied and data-derived weights, our model achieves an 88% overlap with team-selected stories and 74% semantic relatedness with differing choices. This indicates that the model effectively captures team priorities. In cases where the model

diverges, it often makes better decisions, improving value-risk balance and project outcomes. Our interview findings further confirm the model's effectiveness and alignment with team planning.

In terms of speed, the time required to generate a sprint plan depends on the number of user stories and dependencies on the backlog. With precomputed outputs of the trained predictive models, evaluations on an Intel Core i9 with 32GB RAM at 5.8 GHz showed that for 71% of teams (with fewer than 100 stories), computation time ranged from seconds to 2 minutes; for larger backlogs (over 100 stories), computation time ranged from 2 to 6 minutes. This is a significant improvement compared to sprint planning meetings, which typically take hours.

Overall, our model streamlines sprint planning, empowering teams to achieve greater productivity and better project outcomes. This is further confirmed by the majority of teams expressing willingness to use the model in practice, citing improvements in productivity, business alignment, and more informed decision-making.

Interactive support tool. While our model offers substantial benefits, it is designed to support, rather than replace, human judgement. Our interview results indicate that teams prefer to integrate the model with their existing planning methods to maintain control over the process. We recommend an interactive approach that allows teams to adjust objectives and proposed plans as needed. An interactive format also enables the consideration of team composition, availability, and story readiness by gathering team input through forms or backlog tool updates. This flexibility ensures that the model remains aligned with evolving objectives and mitigates concerns about overreliance. Previous research [259, 265] has explored interactive methods for incorporating human expertise into release plan optimization.

Model explainability. Our qualitative evaluation highlights the importance of providing a rationale for story selection to improve model usability. Techniques such as Scenario Discovery [266, 267] and post-optimal analysis (e.g., [268, 269]) can be used to examine how variations in input parameters or assumptions impact the outcomes of an optimization model. This analytical process can help teams uncover patterns among variables and better understand the model's story selection. For example, teams may discover that certain types of stories are prioritized in response to specific project constraints or that story priorities shift based on project urgency.

Implementation effort. Our model learns from past team performance to estimate delay risk and identify patterns in planning behavior, making its insights team-specific. To address data scarcity, especially with new teams [270], we recommend developing a generalized model trained at the product or department level. This approach involves training on historical log data from multiple teams within a product or department, while still allowing individual teams to supply factor weights for model customization. Further analysis at ING shows that product- and department-level models achieve moderate Jaccard similarities (0.62 to 0.68) and cosine similarities (0.59 to 0.63) between proposed and actual sprint plans. These results suggest that generalized models can provide a reasonable baseline for new teams until sufficient team-specific data is available.

5.5.2 Implications for Researchers

Importance of prioritization criteria. Agile methods typically emphasize business value as the main prioritization criterion, a topic that has sparked debate in prior research [107, 108]. Our survey results reveal a more nuanced perspective. We found that the priority of a story is determined by a combination of factors, with urgency, sprint goal alignment, and business value as key drivers. However, the impact of these factors varies across project settings, highlighting the need for contextual support in sprint planning. Future work should examine the influence of project characteristics on prioritization criteria through statistical controls, such as multiple regression analysis. A deeper understanding could inform the redesign or reframing of agile prioritization methods to better align with actual practice.

Strategic alignment and delay risk. In our study, strategic alignment emerged as a new factor affecting story prioritization. Although it ranked lower in survey responses, interview findings emphasize the importance of integrating long-term product strategy into sprint plan generation. While not currently integrated into our model, measuring strategic alignment holds potential for future research. One approach could involve reviewing strategy documents and comparing them with story descriptions to assess alignment. Additionally, our analysis of delay risk highlights its significant impact on story selection for sprint planning. While our current approach estimates delay risk for individual stories, future work could estimate overall delay risk for the sprint plan by considering dependencies among stories and propagation effects. Initial efforts in this direction have been made by Choetkiertikul et al. [21] using network analysis.

Scenario-based modeling. Interview results suggest that using complementary techniques, such as scenario-based modeling, could improve the model's alignment with team planning. Teams frequently adjust their prioritization of user stories in response to events, such as delays or strategic shifts. Future research should focus on identifying scenarios that impact sprint planning, either through direct team input or automated extraction from backlog data. Prior studies by Sutcliffe et al. [271] and Regnell et al. [272] have explored scenario-based modeling for requirements engineering.

5.6 Threats to Validity

Internal validity. We acknowledge that surveys and interviews may contain ambiguous questions and introduce biases [171]. To address this, we used terminology familiar to the case company, ordered questions sequentially, and randomized the order of prioritization criteria [138].To counter social desirability bias [175], we informed participants that the responses would be kept confidential and evaluated in aggregated form. Our survey may have been subject to non-response bias [173], especially among teams struggling to meet sprint commitments.

Construct validity. Poor record keeping may have influenced the data variables we used to measure prioritization criteria and story delay [176]. Story delay is assessed based on the number of sprints a story has been part of, yet inaccuracies may occur if teams close

their stories too early or too late. Although ING encourages teams to deliver on-time, some teams may not take their sprint commitments seriously, adding stories to sprints without intent to deliver. We addressed these concerns by collecting data from a large number of sprints and teams over a four year span.

External validity. The generalizability of findings is an important concern for any single-case study. Although we analyzed data from various teams and products, our findings may not be representative of software projects in other organizations or open-source settings. In other settings, teams may have more dynamic setups and different planning practices. While the high number of teams and availability of historical data may be more common in large-scale organizations, we expect our findings on the prioritization criteria and model impact to be transferable to other settings, regardless of scale. Our approach to generating sprint plans can be applied as is to backlog data from other agile software organizations. It is important to note that ING's strict security regulations as a financial organization may have influenced the prioritization criteria rankings. As a result, these rankings may be more relevant to organizations with similar business- or safety-critical systems. Further research is required to validate our findings in other settings and reach more general conclusions.

5.7 Conclusions

Sprint planning is crucial for the successful execution of agile software projects. While various prioritization criteria influence the selection of user stories for sprint planning, their relative importance remains largely unexplored, especially across diverse project contexts. In this paper, we investigated how prioritization criteria vary across project settings and proposed a context-aware optimization model to generate sprint plans that align with team goals and performance. By integrating teams' planning objectives and sprint history, the model adapts to team contexts, estimating prioritization criteria and identifying patterns in planning behavior. We applied our approach to real-world data from thousands of sprints and evaluated our model through both quantitative and qualitative analyses. The key findings of this study include:

1. Urgency, sprint goal alignment, and business value emerged as the most important prioritization criteria. Their influence varies depending on project characteristics, such as resources, priority, client type, and security level.

2. Our model outperforms the state-of-the-art in aligning with team planning and boosts team performance by generating sprint plans that deliver more business value, align more closely with sprint goals, and better mitigate delay risks.

3. The majority of teams found our approach to be consistent with their goals and valuable as interactive support.

We identified promising areas for future research, including interactive optimization, scenario-based modeling, and model explainability. Progress in these areas is crucial to better understand and support planning practices in agile software development.

6

Conclusion

This chapter revisits the thesis research questions, discusses threats to the overall validity of the thesis, and provides recommendations for agile software organizations. We conclude with future research directions and the broader implications of our findings.

6.1 Research Questions Revisited

In this section, we revisit and answer our thesis research questions outlined in Chapter 1.

TRQ-1: What are the most relevant factors and interactions affecting the on-time delivery of epics?

Our research identifies 25 factors that are perceived to affect the timeliness of epic deliveries. We categorized these factors into five groups: organizational, technical, people, process, and project-related factors. Notably, the most influential factors are predominantly social and organizational, highlighting the critical role of human dynamics in software development. To investigate these influences, we conducted a mixed-methods study, combining expert perceptions with data-driven validation.

To understand how practitioners perceive the causes of delays, we conducted two rounds of surveys with software professionals at ING. In the first round, we collected 298 open-ended responses, identifying 25 factors affecting on-time delivery. In the second round, we surveyed 337 experts, asking them to rank these factors by perceived impact. The most critical factors included requirements refinement, task dependencies, organizational alignment, and organizational politics. To validate these perceptions, we analyzed software repository data from 185 development teams at ING. We quantified and statistically modeled the identified factors, demonstrating that a subset of 13 proxy measures explains 67% of the variance in epic schedule deviations. The most influential predictors included project size, number of task dependencies, historical delivery performance, team familiarity, and developer experience.

Through open-ended survey responses, we identified direct, indirect, and contributory relationships between influential factors and on-time delivery. Some factors exert a direct impact by causing unplanned waiting times, rework, or changes in team effectiveness. Others interact in hierarchical relationships, where organizational factors influence

people-related factors, which in turn affect technical factors. Technical factors are perceived to have a direct effect on delivery timeliness. This layered structure suggests that the impact of technical factors on delays is mediated by the broader organizational and social context.

To represent these relationships, we developed a conceptual framework illustrating the factors and their interactions with on-time delivery. This framework provides a structured approach for organizations to (1) identify and manage delay risks at different levels and (2) formulate actionable pathways to improve delivery performance. Interventions may involve both direct and indirect approaches, addressing factors that have an immediate impact on delivery and those that influence it indirectly through intermediate effects. Direct interventions, such as improving code quality, can lead to immediate benefits by reducing rework. Indirect interventions, such as establishing stronger executive support, can foster a more stable, committed, and skilled workforce, better equipped to address delays caused by technical issues.

Overall Conclusion: Achieving on-time software delivery requires a holistic approach that accounts for the interplay between social and technical factors. Our findings indicate that delays in epic deliveries are primarily driven by human and organizational dynamics rather than purely technical challenges. Addressing these factors through improved organizational alignment, structured refinement processes, and a strong team culture can significantly improve delivery performance. Understanding these interactions and adopting proactive strategies is essential for enhancing software delivery outcomes.

TRQ-2: How can the concept of delay patterns be adapted and applied to continuously predict overall delays in epic deliveries?

Our research explores the application of delay patterns, widely studied in the field of transport, to predicting software project delays. Prior work in transport (e.g., [102, 103]) has demonstrated that delays tend to develop in recurring patterns over time, which can be leveraged for more accurate forecasting of future delays.

To investigate whether similar patterns emerge in software development, we analyzed time series data from 4,040 epics across 270 teams at ING. Each time series captures intermediate delay values recorded at predefined milestones within an epic's timeline. These milestones can be determined based on fixed time intervals (e.g., iteration lengths) or progress-based completion rates (e.g., percentages of planned epic duration). Through clustering analysis, we identified four distinct delay patterns that recur across software projects at ING. These clusters exhibit significant differences in overall delay, demonstrating their relevance as predictive features for ongoing projects and early risk identification.

Agile software development requires effort estimation models that can adapt to evolving project conditions and changes in team performance. However, existing models are static: they estimate effort based on initial predictor values without accounting for temporal variations during project execution. To address this limitation, we developed a dynamic prediction model that continuously updates overall delay estimates using delay patterns and Bayesian inference. Our model detects delay patterns as they emerge and incorporates them into ongoing predictions. The Bayesian framework enables adaptive learning by updating its beliefs based on observed changes in delivery performance and predictor variables over time.

An empirical evaluation shows that incorporating delay patterns significantly improves prediction accuracy. Our approach increases Standardized Accuracy by 9–20% and reduces Mean Absolute Error (MAE) by 19–66%. Compared to static models, our dynamic approach achieves substantial improvements, outperforming state-of-the-art baselines by 12–57% in Standardized Accuracy and 16–81% in MAE. Notably, our model delivers highly accurate predictions even in the early stages of a project (i.e., at 10–30% of its duration), allowing for timely interventions. As the project progresses, the model's predictions become increasingly more certain and precise.

Overall Conclusion: Our results confirm that, as in transport, delays in software projects follow recurrent patterns that can be used to improve forecasting models. More broadly, our findings highlight the advantages of dynamic delay prediction models that continuously adapt to evolving project conditions. By capturing the temporal dynamics of software project delays, our approach provides actionable insights that enable teams to anticipate and mitigate schedule risks throughout the project life cycle.

TRQ-3: Does the use of team features and incremental learning methods improve the accuracy of delay predictions in user stories?

The collaborative nature of agile teams creates complex team dynamics that influence delivery performance. Our research shows that incorporating team-related factors and incremental learning methods improves the accuracy of delay predictions in user stories.

We analyzed backlog data from 571 teams, covering 765,200 user stories, to identify risk factors associated with delays. This analysis revealed 24 risk factors spanning both technical and team-related aspects, with weak to moderate correlations to user story delays. Among these, developer workload, team experience, team stability, and past effort estimates emerged as the most influential team-related factors.

Based on these factors, we developed predictive models using different sets of features to estimate both the likelihood and duration of delays. Our findings show that including team-related features improves delay predictions, with average improvements of 18% in precision, 22% in recall, 20% in F1-score, and 20% in AUC. While both technical and team-related factors contribute to predicting whether a delay will occur, our results indicate that the duration of delays is primarily influenced by team characteristics.

To explore the impact of incremental learning, we evaluated our models using a sliding window approach. This method enables models to continuously learn from recent delivery performance while discarding outdated data. It allows models to adapt to evolving team dynamics. Our results show that training on recent user stories through a sliding window improves prediction accuracy and model robustness. Specifically, this approach improves precision by 6–17%, recall by 5–21%, F1-score by 5–15%, and AUC by 3–8%. The sliding window method is particularly effective in capturing shifts in team performance, leading to more stable and reliable predictions over time.

Overall Conclusion: Overall, our findings suggest that planning in agile settings can be significantly improved by integrating team-related factors and incremental learning methods into analysis and predictive models. By accounting for the evolving nature of teams, these methods offer a more adaptive approach to forecasting delays, supporting more informed decision-making in sprint planning.

TRQ-4: (a) How does the importance of story prioritization criteria vary across project settings?, (b) How can teams' expertise and sprint history be integrated into a model to generate sprint plans that align with team goals and performance?

Agile methods typically emphasize business value as the main criterion for prioritizing user stories, a topic that has sparked debate in previous work [107, 108]. Our research reveals a more nuanced perspective.

Through a survey with 52 teams at ING, we found that story prioritization is determined by a combination of factors, with urgency, sprint goal alignment, and business value as key drivers. The relative weight of these criteria varies across projects, depending on factors such as resource availability, project priority, client type, and security constraints. These findings challenge common assumptions about agile prioritization and highlight the need for contextual support in sprint planning.

To address this need, we developed an optimization model that generates sprint plans tailored to each team's specific goals and historical performance. The model learns from past team performance to identify and incorporate planning behavior patterns. We analyzed historical backlog data from 4,841 sprints and used machine learning techniques to estimate the prioritization criteria for user stories. To capture teams' planning behavior, we built a predictive model that learns from a team's sprint history and estimates the likelihood of a story being selected for an upcoming sprint. We obtained team-provided planning objectives and prioritization weights from survey responses. Combining these elements, we designed an optimization model that adapts to each team's context, producing sprint plans aligned with their goals and past performance.

We assessed our model through both quantitative and qualitative evaluations. The quantitative evaluation demonstrates significant improvements in team alignment and sprint plan effectiveness. The model achieves an 88% overlap in selected stories with teams' actual sprint plans and a 74% semantic relatedness between differing stories. Our model outperforms the state-of-the-art and improves team performance by generating sprint plans that deliver 29% more business value, exhibit 14% stronger alignment with sprint goals, and reduce delay risk by 42%. The qualitative evaluation, based on team interviews, confirmed the model's usability and practical value. Most teams found the approach to be well-aligned with their decision-making processes and beneficial as interactive support.

Overall Conclusion: Overall, our results demonstrate that the efficiency and effectiveness of sprint planning can be significantly improved through the use of context-aware optimization methods.

6.2 Threats to Validity

In this section, we discuss three overarching threats to validity that may have influenced the conclusions of this thesis. These threats summarize and complement the more specific validity concerns addressed in each study's respective chapter.

- **Construct Validity**
  In our analysis of project repository data, we consider data variables as proxies for measuring delays and risk factors. However, this approach introduces potential threats to construct validity due to measurement errors [176]. The variables we use may not fully capture the intended meaning of the concepts or constructs. Certain

factors, such as team commitment and organizational alignment, are quantifiable in principle but not directly measurable. To map proxy variables to risk factors, we had to find acceptable trade-offs between the precision of these proxies and the availability of data at ING. We acknowledge that for some factors, more precise alternatives can be found in related work. Furthermore, poor record keeping may have influenced the accuracy of data variables. For example, we measure delivery delays based on deviations between planned and actual completion dates in backlog management data. However, it might happen that teams close their tasks too early or too late. While all teams at ING are encouraged to deliver on-time, some may not treat their delivery deadlines as seriously as others, potentially adding work items to their planning without the commitment to deliver on-time. To mitigate these threats, we collected real-world data from a large number of deliveries and teams over several years at ING, aiming to reduce potential biases and inaccuracies.

- **Internal Validity**
  One key challenge in our predictive modeling is class imbalance, as the majority of deliveries in our dataset are non-delayed. This imbalance can hinder the model's ability to effectively learn to identify late deliveries. To address this issue, we applied weighted variants of performance metrics and used the Area Under the Curve (AUC) score, which is insensitive to class imbalance. Additionally, we validated our assumptions through statistical testing [191] and adhered to best practices in evaluating effort estimation models [39, 53, 241]. However, we acknowledge that more advanced techniques, such as statistical oversampling [242], could be used. Another potential threat to our internal validity is that the patterns observed in the training data may not accurately reflect those in the test data. This discrepancy could arise due to structural changes in teams or management over time. To mitigate this risk, we employed time-based cross-validation instead of traditional random data splitting. This approach better simulates real-world prediction scenarios by preserving the chronological order of data.

- **External Validity**
  As with any single-case empirical study, external threats are concerned with our ability to generalize our results beyond the studied context [172]. ING is a large-scale organization with thousands of developers working across diverse domains, including banking applications, cloud software, and software tools. While we analyzed a large number of projects and teams to capture variations, our findings may not be representative of software projects in other organizations or open-source settings. Team structures, planning practices, and organizational cultures can vary significantly across contexts, potentially influencing the applicability of our results. Moreover, ING's strict security regulations and risk-averse culture may have influenced some of the identified risk factors. In financial organizations, failure in certain business-critical systems is unacceptable, which may have affected the prioritization of risk factors. As a result, our findings are more likely to generalize to software organizations operating in regulated industries with similar security and compliance constraints. Additionally, we identified factors related to organizational fragmen-

tation, such as dependency management and organizational alignment, which may be more common in large-scale organizations.

Although we aimed for generality, our findings remain specific to the context and time period studied. Further empirical validation is necessary to determine the extent to which these results transfer to other settings. However, our findings align with prior research on effort estimation and risk management, suggesting that the risk factors we identified are transferable to some extent to other settings. However, specific results regarding the ranking of factors, their relationships, and statistical outcomes are bound to the scale and context of ING. It is important to note that the influence of factors on on-time delivery may vary depending on the scale and context of development work. Therefore, replicating this research in different settings is required to confirm our findings and draw more general conclusions.

6.3 Recommendations for Software Organizations

In this section, we provide recommendations (R1–R5) based on our research findings to help agile software organizations enhance effort estimation and project planning. These recommendations offer actionable strategies for improving delivery performance.

R1: A socio-technical approach is essential for improving on-time delivery and sprint planning. Strategies for improving on-time delivery must account for the complex interactions between social and technical factors. Our conceptual framework offers practitioners a comprehensive overview of influential factors and proxy measures that should be analyzed to develop models that improve on-time delivery. An assessment of the most significant factors would be a good starting point for further exploring influential factors in other settings. While the framework does not establish causal relationships, it provides a foundation for forming testable hypotheses about the mechanisms driving delays. By systematically testing these hypotheses, software organizations can identify root causes, develop targeted interventions, and refine their strategies for mitigating delivery risks.

In Chapter 5, we show that the efficiency and outcomes of sprint planning can be significantly improved through context-aware optimization methods. These methods combine social elements (e.g., team expertise and collaboration) with technical elements (e.g., story risk and affinity) to generate sprint plans that better align with team objectives. Our approach leads to significant improvements, including 29% more business value delivered per sprint, 14% stronger alignment with sprint goals, 42% reduction in delay risk. The strong alignment between model-generated and team-selected sprint plans indicates that our approach effectively captures team priorities and preferences. Moreover, in cases where the model deviates from team decisions, it often suggests better alternatives, resulting in a more balanced trade-off between value and risk.

R2: Data-driven techniques enhance factor identification and predictive accuracy. Our research shows that integrating data-driven techniques with expert judgment improves both the identification of influential factors and the accuracy of delay predictions. Expert-based approaches capture domain knowledge and contextual experience, while data-driven methods detect underlying patterns that may not be immediately apparent.

Since each method identifies only a partially overlapping set of relevant factors, combining both provides a more comprehensive understanding of delay risks. By systematically collecting and analyzing these factors, organizations can identify early warning signs of potential delays.

For delay prediction, our findings show that data-driven models significantly improve predictive accuracy for both short- and long-term estimates. In short-term delay prediction for user stories, incorporating team-related factors into predictive models significantly improves accuracy. Incremental learning approaches, such as sliding windows, enable models to continuously adapt to team dynamics and external changes, including unexpected bugs and incidents. In long-term delay prediction for epics, leveraging delay patterns improves accuracy by 9–20% and reduces Mean Absolute Error by 19–66%. While delay patterns may vary across organizations, their predictive power increases when combined with dynamic prediction models that incorporate changes during project execution. This combined approach leads to more accurate and reliable schedule estimates over time, enabling teams to detect risks early and respond proactively throughout the development life cycle.

Although predicting delays in later project stages (beyond 50% project completion) may offer more certain estimates, it also limits opportunities for corrective action. Our research shows that the optimal trade-off between prediction time and accuracy occurs within the first 10–30% of project duration, when early intervention is most effective. Within this window, our dynamic model consistently outperforms static alternatives, achieving 12–29% higher Standardized Accuracy and 16–41% lower Mean Absolute Error.

R3: Developing tailored estimation and planning models improves predictive accuracy and project outcomes. In the pursuit of the best effort estimation model, extensive research has compared different approaches, yet no single method consistently outperforms others across all settings. This variability largely stems from the dynamic nature of factors affecting delays in software projects. Our findings indicate that these factors vary significantly depending on the development context, emphasizing the need for estimation models that are tailored to specific teams and projects. Additionally, the large differences in productivity among developers and teams further highlight the importance of models that effectively account for these variations.

Most existing estimation models primarily rely on metadata and textual features of software tasks. By leveraging historical backlog data and incorporating the influential factors identified in this thesis, organizations can tailor their models to their specific context. This approach provides deeper and more actionable insights into their development processes. Similarly, our research reveals that the criteria for prioritizing and selecting user stories for sprint planning vary across different project settings. This highlights the need for context-aware planning models that integrate team goals and historical performance to optimize decision-making. Models that automate agile planning should adapt to team-specific contexts rather than applying generic prioritization strategies.

Overall, our findings suggest that software organizations should develop customized estimation and planning models, rather than relying on generic tools, to ensure accuracy within their specific context.

R4: Historical delivery performance should be used to estimate effort intervals, rather than relying on single-point estimates. Our research highlights the benefits of using probabilistic effort estimates, expressed as effort intervals, over traditional point estimates. Effort intervals provide information about the uncertainty of an estimate and can help organizations increase confidence in project planning. Instead of relying on expert judgment to define minimum and maximum effort, software organizations should base these estimates on historical delivery performance data.

Our study on dynamic delay prediction (Chapter 3) demonstrates the effectiveness of Bayesian methods in quantifying and updating uncertainty of predictions over time. Unlike other models, Bayesian approaches provide probability distributions that represent the credibility of an estimate, enabling teams to make more informed planning decisions.

R5: Automated support for effort estimation and planning should complement human judgment and current practices. The models developed in this thesis should be used as decision support systems that complement, rather than replace, human judgment. Automated predictions and planning recommendations provide valuable insights, but teams must interpret them within context to ensure alignment with current priorities, constraints, and strategic goals.

While delay prediction models leverage historical data to enhance accuracy, they may not fully capture unforeseen organizational shifts or emergent risks. Human judgment is essential for context-aware decision-making, ensuring that model outputs are critically assessed and adjusted based on real-time project conditions. As the models developed in this thesis generate team-specific predictions, they can serve as feedback mechanisms that increase awareness of behavioral patterns during estimation and planning. This feedback can encourage more accurate effort estimates and foster productive behavior changes.

For sprint planning, automation streamlines prioritization and reduces the effort required to manually select stories. However, agile planning remains a collaborative process that depends on discussions, shared understanding, and collective decision-making. Trust in model recommendations is essential for adoption; keeping humans in the loop allows teams to validate, challenge, and refine model outputs based on their domain knowledge. To enhance transparency and usability, organizations should integrate explainability techniques such as Scenario Discovery [266, 267] and post-optimal analysis [268, 269] to help teams interpret model recommendations more effectively.

6.4 Conclusion and Future Work

In this section, we summarize our main findings and outline directions for future research.

Social and organizational factors play a central role in on-time delivery. Our research has identified new factors that influence the timely delivery of epics (Chapter 2) and user stories (Chapter 4). While prior studies [154, 220] have shown that team familiarity and agile maturity enhance team performance, their specific impact on on-time delivery had not been explored before. Our findings confirm that these factors significantly contribute to on-time epic completion, with agile maturity ranking among the top 10 most influential factors. This suggests that team familiarity and agile maturity improve delivery predictability, helping teams navigate challenges more effectively.

Additionally, we found that task dependencies and organizational alignment are among the most influential factors affecting schedule deviations. Their impact suggests that the organizational environment plays a more substantial role than previously acknowledged. This highlights the need for further exploration across diverse development contexts to better understand how these factors interact in different organizational settings.

At the user story level, we found that developer workload, team experience, team stability, and past effort estimates significantly affect delivery timelines. Future research should examine these team-related factors in different organizational contexts to assess their broader implications for software development performance.

Understanding delay patterns enables proactive risk management. To improve the implementation of delay countermeasures, it is essential to better understand the root causes of delays and how delay patterns emerge. Our study on dynamic prediction characterizes these patterns in terms of risk factors, revealing significant differences. Although we cannot directly infer causal relationships between risk factors and delay patterns, factor analysis enables the formulation of hypotheses about potential causes. Systematically testing these hypotheses could lead to actionable insights and inform more effective mitigation strategies. Future work should apply causal inference methods, such as Causal Discovery [203], to identify the underlying causes of delay trends and fluctuations. Additionally, investigating the propagation of delays across interdependent software deliveries could offer valuable insights into how schedule deviations escalate across projects.

Monitoring social dynamics improves estimation and planning accuracy. Our research proposes a set of new methods for monitoring social-driven factors and team dynamics in software development. Using historical backlog and project data, we derive statistics at both the team and individual levels to quantify aspects of team behavior that were previously difficult to capture [239, 240]. Given the strong influence of social factors on delays, future research should develop new approaches to systematically monitor and model team dynamics, including trust, leadership, cohesion, and communication. Integrating team behavioral insights into effort estimation and planning models could enhance their overall predictive power.

Agile project management tools should incorporate predictive analytics. Our predictive models can serve as feedback mechanisms that help teams recognize behavioral patterns during planning. By increasing awareness of planning biases, these models can encourage productive behavior changes and facilitate more accurate schedule estimates. However, to fully realize the benefits of predictive modeling, future research should explore how delay prediction models can be embedded into agile project management tools. Existing tools primarily track and manage project artifacts but lack advanced analytics to extract actionable insights for planning.

Integrating actionable insights on team dynamics and sprint-level risks would enable teams to make more informed planning decisions, ultimately leading to better project outcomes. Enhancing these tools with predictive capabilities, such as scenario-based modeling and multi-team planning support, presents promising directions for future research. Additionally, concerns raised in our interviews, including the ordering of story imple-

mentation and alignment with portfolio strategies, should be addressed to optimize the usability and effectiveness of these tools.

Agile prioritization methods should account for context-specific variations. Our findings challenge traditional agile prioritization methods, which predominantly emphasize business value as the main prioritization criterion. Chapter 5 demonstrates that story prioritization is not driven by a single criterion but rather a combination of factors, with urgency, sprint goal alignment, and business value as key drivers. The relative importance of these factors varies depending on project characteristics such as resource availability, project priority, client type, and security level.

These findings suggest that agile prioritization methods should be context-aware, rather than applying one-size-fits-all strategies. Future research should investigate how project characteristics influence prioritization criteria using statistical controls such as multiple regression analysis. A deeper understanding of prioritization decisions could inform the redesign of agile prioritization frameworks to better reflect real-world planning practices.

A step toward a relational theory of on-time software delivery. Our research lays the groundwork for a relational theory of on-time delivery by highlighting the hierarchical relationships between social and technical factors. Our conceptual framework suggests that the social context of development work determines how technical factors influence delivery timelines. We anticipate that this framework, along with the set of factors and proxy measures identified in this thesis, will serve as a foundation for future research on software delivery performance. While primarily descriptive, the framework also aims to specify the relationships between factors and provide an initial explanation of their interactions. The hypotheses for corrective actions proposed in Section 2.3 outline potential research directions for further refinement.

To validate and extend this framework, future studies should replicate our mixed-methods approach across diverse organizational contexts. Investigating on-time delivery in companies of varying sizes, structures, and industry domains could reveal additional influential factors and improve our understanding of existing relationships. Furthermore, examining organizations with different levels of management support, stability, and agile maturity could shed light on how social and organizational dynamics affect delivery performance.

A key challenge for future research is to move beyond descriptive analysis and establish a more formal relational theory by identifying the causal mechanisms behind delays. Applying causal inference techniques, such as time-series analysis or controlled experiments with development teams, could reveal how and why certain factors influence delivery timelines. A more comprehensive relational theory would provide actionable insights for project managers, enabling them to design effective risk management strategies and targeted interventions that enhance on-time delivery performance.

6.5 Implications and Outlook

In this thesis, we have investigated the factors that influence on-time delivery in agile software development. Our findings highlight the complex interplay between social and technical factors, emphasizing the need for context-aware approaches to effort estimation

and planning. To improve delivery performance, we developed predictive and planning models that integrate team-related factors, leverage dynamic learning techniques, and incorporate delay patterns. These models provide actionable strategies for enhancing delivery predictability and risk management.

Our research offers valuable contributions to both industry and academia. For practitioners, it provides practical insights into managing delay risks, improving effort estimation, and optimizing sprint planning. By identifying key delay drivers and their interactions, our work enables organizations to take a more proactive approach to mitigating risks in large-scale agile settings. The models developed in this thesis can inform the design of automated planning support tools, helping teams anticipate delays and make more informed planning decisions. Additionally, integrating team expertise and historical performance into planning models offers a structured approach to balancing competing priorities and improving delivery outcomes.

For academia, this work advances the understanding of delay factors in large-scale agile development, bridging gaps between effort estimation, planning, and team dynamics. Our conceptual framework and predictive models contribute to the broader development of a relational theory of on-time software delivery. They provide a foundation for future research on causal relationships underlying software delays. By demonstrating how data-driven techniques can improve estimation accuracy and decision-making, this thesis opens new avenues for research in predictive analytics for agile project management.

Ultimately, our findings can help agile software development become more predictable. By refining estimation and planning methods, integrating predictive analytics into agile workflows, and further exploring the socio-technical dynamics of software delivery, both practitioners and researchers can drive the field forward. This research takes a step toward more reliable and data-driven project management, giving teams the insights and tools they need to navigate the uncertainties of modern software development.

Bibliography

References

[1] Bent Flyvbjerg and Alexander Budzier. Why your it project might be riskier than you think. *arXiv preprint arXiv:1304.0265*, 2013.

[2] Torleif Halkjelsvik and Magne Jørgensen. From origami to software development: A review of studies on judgment-based predictions of performance time. *Psychological bulletin*, 138(2):238, 2012.

[3] Magne Jørgensen. What we do and don't know about software development effort estimation. *IEEE software*, 31(2):37–40, 2014.

[4] Michael Bloch, Sven Blumberg, and Jürgen Laartz. Delivering large-scale it projects on time, on budget, and on value. *Harvard Business Review*, 5(1):2–7, 2012.

[5] Fred J Heemstra. Software cost estimation. *Information and software technology*, 34(10):627–639, 1992.

[6] Magne Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, 2004.

[7] Bent Flyvbjerg, Alexander Budzier, Jong Seok Lee, Mark Keil, Daniel Lunn, and Dirk W Bester. The empirical reality of it project cost overruns: Discovering a power-law distribution. *Journal of Management Information Systems*, 39(3):607–639, 2022.

[8] Júlio Menezes, Cristine Gusmão, and Hermano Moura. Risk factors in software development projects: a systematic literature review. *Software Quality Journal*, 27(3):1149–1174, 2019.

[9] Tsun Chow and Dac-Buu Cao. A survey study of critical success factors in agile software projects. *Journal of systems and software*, 81(6):961–971, 2008.

[10] Muhammad Usman, Emilia Mendes, and Jürgen Börstler. Effort estimation in agile software development: a survey on the state of the practice. In *Proceedings of the 19th international conference on Evaluation and Assessment in Software Engineering*, pages 1–10, 2015.

[11] Alistair Cockburn and Jim Highsmith. Agile software development, the people factor. *Computer*, 34(11):131–133, 2001.

[12] Muhammad Usman, Emilia Mendes, Francila Weidt, and Ricardo Britto. Effort estimation in agile software development: a systematic literature review. In *Proceedings of the 10th international conference on predictive models in software engineering*, pages 82–91. ACM, 2014.

[13] Mike Cohn. *Agile estimating and planning*. Pearson Education, 2005.

[14] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[15] Dean Leffingwell. *Scaling software agility: best practices for large enterprises*. Pearson Education, 2007.

[16] Emanuel Dantas, Mirko Perkusich, Ednaldo Dilorenzo, Danilo FS Santos, Hyggo Almeida, and Angelo Perkusich. Effort estimation in agile software development: an updated review. *International Journal of Software Engineering and Knowledge Engineering*, 28(11n12):1811–1831, 2018.

[17] Adam Trendowicz and Jürgen Münch. Factors influencing software development productivity—state-of-the-art and industrial experiences. *Advances in computers*, 77:185–241, 2009.

[18] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Pham, Aditya Ghose, and Tim Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7):637–656, 2018.

[19] Simone Porru, Alessandro Murgia, Serge Demeyer, Michele Marchesi, and Roberto Tonelli. Estimating story points from issue reports. In *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 1–10, 2016.

[20] Ezequiel Scott and Dietmar Pfahl. Using developers' features to estimate story points. In *Proceedings of the 2018 International Conference on Software and System Process*, pages 106–110, 2018.

[21] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Predicting delays in software projects using networked classification (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 353–364. IEEE, 2015.

[22] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Predicting the delay of issues with due dates in software projects. *Empirical Software Engineering*, 22(3):1223–1263, 2017.

[23] Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: an empirical study of commercial software projects. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 1042–1051. IEEE, 2013.

[24] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Aditya Ghose, and John Grundy. Predicting delivery capability in iterative software development. *IEEE Transactions on Software Engineering*, 44(6):551–573, 2017.

[25] Pekka Abrahamsson, Raimund Moser, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. Effort prediction in iterative software development processes–incremental versus global prediction models. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 344–353. IEEE, 2007.

[26] Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15:654–693, 2010.

[27] Andrew Begel and Nachiappan Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 255–264. IEEE, 2007.

[28] Laurie Williams and Alistair Cockburn. Agile software development: It's about feedback and change. *Computer*, 36(6):39–43, 2003.

[29] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.

[30] VersionOne. 16th State of Agile Survey. `https://info.digital.ai/rs/981-LQX-968/images/SOA16.pdf`, .

[31] Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time*. Crown Currency, 2014.

[32] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.

[33] John Erickson, Kalle Lyytinen, and Keng Siau. Agile modeling, agile software development, and extreme programming: the state of research. *Journal of Database Management (JDM)*, 16(4):88–100, 2005.

[34] Kieran Conboy and Noel Carroll. Implementing large-scale agile frameworks: challenges and recommendations. *IEEE Software*, 36(2):44–50, 2019.

[35] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[36] Maria Paasivaara. Adopting safe to scale agile in a globally distributed organization. In *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pages 36–40. IEEE, 2017.

[37] Maria Paasivaara, Benjamin Behm, Casper Lassenius, and Minna Hallikainen. Large-scale agile transformation at ericsson: a case study. *Empirical Software Engineering*, 23(5):2550–2596, 2018.

[38] H Frank Cervone. Understanding agile project management methods using scrum. *OCLC Systems & Services: International digital library perspectives*, 2011.

[39] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, 32(11):883–895, 2006.

[40] Magne Jørgensen. A critique of how we measure and interpret the accuracy of software development effort estimation. In *First International Workshop on Software Productivity Analysis and Cost Estimation. Information Processing Society of Japan, Nagoya*. Citeseer, 2007.

[41] Adam Trendowicz and Ross Jeffery. Software project effort estimation. *Foundations and Best Practice Guidelines for Success, Constructive Cost Model–COCOMO pags*, pages 277–293, 2014.

[42] Magne Jørgensen and Stein Grimstad. Avoiding irrelevant and misleading information when estimating development effort. *IEEE software*, 25(3):78–83, 2008.

[43] Viljan Mahnič and Tomaž Hovelja. On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086–2095, 2012.

[44] Carlos Joaquín Torrecilla-Salinas, Jorge Sedeño, MJ Escalona, and Manuel Mejías. Estimating, planning and managing agile web development projects under a value-based perspective. *Information and Software Technology*, 61:124–144, 2015.

[45] Pekka Abrahamsson, Ilenia Fronza, Raimund Moser, Jelena Vlasenko, and Witold Pedrycz. Predicting development effort from user stories. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 400–403. IEEE, 2011.

[46] Danh Nguyen-Cong and De Tran-Cao. A review of effort estimation studies in agile, iterative and incremental software development. In *The 2013 RIVF International Conference on Computing & Communication Technologies-Research, Innovation, and Vision for Future (RIVF)*, pages 27–30. IEEE, 2013.

[47] Simon Grapenthin, Steven Poggel, Matthias Book, and Volker Gruhn. Facilitating task breakdown in sprint planning meeting 2 with an interaction room: an experience report. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 1–8. IEEE, 2014.

[48] Sungjoo Kang, Okjoo Choi, and Jongmoon Baik. Model-based dynamic cost estimation and tracking method for agile software development. In *2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, pages 743–748. IEEE, 2010.

[49] Irum Inayat, Siti Salwah Salim, Sabrina Marczak, Maya Daneva, and Shahaboddin Shamshirband. A systematic literature review on agile requirements engineering practices and challenges. *Computers in human behavior*, 51:915–929, 2015.

[50] Manish Agrawal and Kaushal Chari. Software effort, quality, and cycle time: A study of cmm level 5 projects. *IEEE Transactions on software engineering*, 33(3):145–156, 2007.

[51] Magne Jørgensen and Tanja M Gruschke. The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *IEEE Transactions on Software Engineering*, 35(3):368–383, 2009.

[52] Adam Trendowicz, Michael Ochs, Axel Wickenkamp, Jürgen Münch, Yasushi Ishigai, and Takashi Kawaguchi. Integrating human judgment and data analysis to identify factors influencing software development productivity. *e-Informatica*, 2(1):47–69, 2008.

[53] Ekrem Kocaguneli, Tim Menzies, and Jacky W Keung. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering*, 38(6):1403–1416, 2011.

[54] Panagiotis Sentas, Lefteris Angelis, Ioannis Stamelos, and George Bleris. Software productivity and effort prediction with ordinal regression. *Information and software technology*, 47(1):17–29, 2005.

[55] Magne Jørgensen, Ulf Indahl, and Dag Sjøberg. Software effort estimation by analogy and "regression toward the mean". *Journal of Systems and Software*, 68(3):253–262, 2003.

[56] Aditi Panda, Shashank Mouli Satapathy, and Santanu Kumar Rath. Empirical validation of neural network models for agile software effort estimation based on story points. *Procedia Computer Science*, 57:772–781, 2015.

[57] Ali Bou Nassif, Mohammad Azzeh, Ali Idri, and Alain Abran. Software development effort estimation using regression fuzzy models. *Computational intelligence and neuroscience*, 2019, 2019.

[58] S Kanmani, Jayabalan Kathiravan, S Senthil Kumar, and Mourougane Shanmugam. Neural network based effort estimation using class points for oo systems. In *2007 International Conference on Computing: Theory and Applications (ICCTA'07)*, pages 261–266. IEEE, 2007.

[59] Satish Kumar, B Ananda Krishna, and Prem S Satsangi. Fuzzy systems and neural networks in software engineering project management. *Applied Intelligence*, 4:31–52, 1994.

[60] Emilia Mendes and Nile Mosley. Bayesian network models for web effort prediction: a comparative study. *IEEE Transactions on Software Engineering*, 34(6):723–737, 2008.

[61] Peter Hearty, Norman Fenton, David Marquez, and Martin Neil. Predicting project velocity in xp using a learning dynamic bayesian network model. *IEEE Transactions on Software Engineering*, 35(1):124–137, 2008.

[62] Federica Sarro, Alessio Petrozziello, and Mark Harman. Multi-objective software effort estimation. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 619–630. IEEE, 2016.

[63] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Trang Thi Minh Pham, Aditya Ghose, and Tim Menzies. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 2018.

[64] Jianfeng Wen, Shixian Li, Zhiyong Lin, Yong Hu, and Changqin Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[65] Pinkashia Sharma and Jaiteg Singh. Systematic literature review on software effort estimation using machine learning approaches. In *2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS)*, pages 43–47. IEEE, 2017.

[66] Lucas D Panjer. Predicting eclipse bug lifetimes. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, pages 29–29. IEEE, 2007.

[67] Pamela Bhattacharya and Iulian Neamtiu. Bug-fix time prediction models: can we do better? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 207–210, 2011.

[68] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, 32(2):69–82, 2006.

[69] Saïd Assar, Markus Borg, and Dietmar Pfahl. Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy. *Empirical Software Engineering*, 21(4):1437–1475, 2016.

[70] Chandra Maddila, Chetan Bansal, and Nachiappan Nagappan. Predicting pull request completion time: a case study on large scale cloud services. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 874–882, 2019.

[71] Ana Magazinius, Sofia Börjesson, and Robert Feldt. Investigating intentional distortions in software cost estimation–an exploratory study. *Journal of Systems and Software*, 85(8):1770–1781, 2012.

[72] Ana Magazinius and Robert Feldt. Confirming distortional behaviors in software cost estimation practice. In *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 411–418. IEEE, 2011.

[73] Magne Jørgensen. The influence of selection bias on effort overruns in software development projects. *Information and Software Technology*, 55(9):1640–1650, 2013.

[74] Magne Jørgensen and Stein Grimstad. Over-optimism in software development projects:" the winner's curse". In *15th International Conference on Electronics, Communications and Computers (CONIELECOMP'05)*, pages 280–285. IEEE, 2005.

[75] Jim Highsmith. *Agile project management: creating innovative products*. Pearson education, 2009.

[76] Julian M Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and software technology*, 75:1–16, 2016.

[77] James Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3:22–23, 2002.

[78] James G March and Zur Shapira. Managerial perspectives on risk and risk taking. *Management science*, 33(11):1404–1418, 1987.

[79] Karel De Bakker, Albert Boonstra, and Hans Wortmann. Does risk management contribute to it project success? a meta-analysis of empirical evidence. *International Journal of Project Management*, 28(5):493–503, 2010.

[80] James J Jiang, Gary Klein, and Thomas L Means. Project risk impact on software development team performance. *Project Management Journal*, 31(4):19–26, 2000.

[81] James J Jiang, Gary Klein, and Richard Discenza. Information system success as impacted by risks and development strategies. *IEEE transactions on Engineering Management*, 48(1):46–55, 2001.

[82] Wen-Ming Han and Sun-Jen Huang. An empirical analysis of risk components and performance on software projects. *Journal of Systems and Software*, 80(1):42–50, 2007.

[83] Linda Wallace, Mark Keil, and Arun Rai. Understanding software project risk: a cluster analysis. *Information & management*, 42(1):115–125, 2004.

[84] Barry W. Boehm. Software risk management: principles and practices. *IEEE software*, 8(1):32–41, 1991.

[85] Marvin J Carr, Suresh L Konda, Ira Monarch, F Carol Ulrich, and Clay F Walker. Taxonomy-based risk identification. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1993.

[86] Robert N Charette. *Software engineering risk analysis and management*. Intertext Publications New York, 1989.

[87] Capers Jones. *Assessment and control of software risks*. Yourdon Press, 1994.

[88] Tom Addison and Seema Vallabh. Controlling software project risks: an empirical study of methods used by experienced project managers. In *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 128–140. Citeseer, 2002.

[89] Henri Barki, Suzanne Rivard, and Jean Talbot. Toward an assessment of software development risk. *Journal of management information systems*, 10(2):203–225, 1993.

[90] Roy Schmidt, Kalle Lyytinen, Mark Keil, and Paul Cule. Identifying software project risks: An international delphi study. *Journal of management information systems*, 17(4):5–36, 2001.

[91] Kweku Ewusi-Mensah. *Software development failures*. Mit Press, 2003.

[92] Sirkka L Jarvenpaa and Blake Ives. Executive involvement and participation in the management of information technology. *MIS quarterly*, pages 205–227, 1991.

[93] Michiel Van Genuchten. Why is software late? an empirical study of reasons for delay in software development. *IEEE Transactions on software engineering*, 17(6):582–590, 1991.

[94] Henri Barki and Jon Hartwick. Rethinking the concept of user involvement. *MIS quarterly*, pages 53–63, 1989.

[95] Hooman Hoodat and Hassan Rashidi. Classification and analysis of risks in software engineering. *World Academy of Science, Engineering and Technology*, 56(32):446–452, 2009.

[96] Janne Ropponen and Kalle Lyytinen. Components of software development risk: How to address them? a project manager survey. *IEEE transactions on software engineering*, 26(2):98–112, 2000.

[97] Linda Wallace, Mark Keil, and Arun Rai. How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model. *Decision sciences*, 35(2):289–321, 2004.

[98] Emmanuel Letier, David Stefan, and Earl T Barr. Uncertainty, risk, and information value in software requirements and architecture. In *Proceedings of the 36th International Conference on Software Engineering*, pages 883–894, 2014.

[99] Yong Hu, Xiangzhou Zhang, EWT Ngai, Ruichu Cai, and Mei Liu. Software project risk analysis using bayesian networks with causality constraints. *Decision Support Systems*, 56:439–449, 2013.

[100] Harry Raymond Joseph. Poster: Software development risk management: using machine learning for generating risk prompts. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 833–834. IEEE, 2015.

[101] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Characterization and prediction of issue-related risks in software projects. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 280–291. IEEE, 2015.

[102] Mehmet Şirin Artan and İsmail Şahin. Exploring patterns of train delay evolution and timetable robustness. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11205–11214, 2021.

[103] Ping Huang, Thomas Spanninger, and Francesco Corman. Enhancing the understanding of train delays with delay evolution pattern discovery: A clustering and bayesian network approach. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):15367–15381, 2022.

[104] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.

[105] Martin Hoegl and Hans Georg Gemuenden. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization science*, 12(4):435–449, 2001.

[106] Yngve Lindsjørn, Dag IK Sjøberg, Torgeir Dingsøyr, Gunnar R Bergersen, and Tore Dybå. Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, 122:274–286, 2016.

[107] Kai Petersen and Claes Wohlin. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of systems and software*, 82(9):1479–1490, 2009.

[108] Zornitza Racheva, Maya Daneva, Klaas Sikkel, Andrea Herrmann, and Roel Wieringa. Do we know enough about requirements prioritization in agile projects: insights from a case study. In *2010 18th IEEE International Requirements Engineering Conference*, pages 147–156. IEEE, 2010.

[109] Alan Moran. Agile risk management. In *Agile Risk Management*, pages 33–60. Springer, 2014.

[110] Breno Gontijo Tavares, Carlos Eduardo Sanches da Silva, and Adler Diniz de Souza. Practices to improve risk management in agile projects. *International Journal of Software Engineering and Knowledge Engineering*, 29(03):381–399, 2019.

[111] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13:303–337, 2008.

[112] Barry Boehm, Hans Dieter Rombach, and Marvin V Zelkowitz. *Foundations of empirical software engineering: the legacy of Victor R. Basili.* Springer Science & Business Media, 2005.

[113] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.

[114] Kathy Charmaz, Liska Belgrave, et al. Qualitative interviewing and grounded theory analysis. *The SAGE handbook of interview research: The complexity of the craft*, 2:347–365, 2012.

[115] Ji Young Cho and Eun-Hee Lee. Reducing confusion about grounded theory and qualitative content analysis: Similarities and differences. *Qualitative Report*, 19(32), 2014.

[116] Joanna F DeFranco and Phillip A Laplante. A content analysis process for qualitative software engineering research. *Innovations in Systems and Software Engineering*, 13(2):129–141, 2017.

[117] Christian Bird, Tim Menzies, and Thomas Zimmermann. *The art and science of analyzing software data*. Elsevier, 2015.

[118] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[119] Elvan Kula, Eric Greuter, Arie van Deursen, and Georgios Gousios. Supplemental material for Factors Affecting On-time Delivery in Large-Scale Agile Software Development. March 2021, Zenodo. https://doi.org/10.5281/zenodo.11625946.

[120] Elvan Kula, Eric Greuter, Arie Van Deursen, and Gousios Georgios. Supplemental material for Dynamic Prediction of Delays in Software Projects Using Bayesian Modeling. June 2023, Zenodo. https://doi.org/10.5281/zenodo.11625842.

[121] Elvan Kula, Arie Van Deursen, and Gousios Georgios. Supplemental material for Modeling Team Dynamics for the Characterization and Prediction of Delays in User Stories. April 2021, Zenodo. https://doi.org/10.5281/zenodo.12206605.

[122] Elvan Kula, Arie Van Deursen, and Gousios Georgios. Supplemental material for Context-Aware Automated Sprint Plan Generation for Agile Software Development. June 2024, Zenodo. https://doi.org/10.5281/zenodo.11522834.

[123] ING Group N.V. Annual Report of 2023. 2024. https://www.ing.com/MediaEditPage/2023-ING-Groep-N.V.-annual-report.htm.

[124] Henrik Kniberg and Anders Ivarsson. Scaling agile@ spotify with tribes, squads, chapters & guilds. *Entry posted November*, 12, 2012.

[125] Mashal Alqudah and Rozilawati Razali. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6):828–837, 2016.

[126] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[127] SonarSource. SonarQube - Continuous Code Quality. 2024. https://www.sonarsource.com/products/sonarqube/.

[128] OpenText. Fortify Static Code Analyzer. 2024. https://www.opentext.com/products/fortify-static-code-analyzer.

[129] Elvan Kula, Eric Greuter, Arie Van Deursen, and Georgios Gousios. Factors affecting on-time delivery in large-scale agile software development. *IEEE Transactions on Software Engineering*, 48(9):3573–3592, 2021. Open Access version: https://pure.tudelft.nl/ws/portalfiles/portal/134949880/Factors_Affecting_On_Time_Delivery_in_Large_Scale_Agile_Software_Development.pdf.

[130] Elvan Kula, Eric Greuter, Arie van Deursen, and Georgios Gousios. Dynamic prediction of delays in software projects using delay patterns and bayesian modeling. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1012–1023, 2023. Open Access version: https://pure.tudelft.nl/ws/portalfiles/portal/180826462/3611643.3616328.pdf.

[131] Elvan Kula, Arie van Deursen, and Georgios Gousios. Modeling team dynamics for the characterization and prediction of delays in user stories. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 991–1002. IEEE, 2021. Open Access version: https://pure.tudelft.nl/ws/portalfiles/portal/105127779/main.pdf.

[132] Elvan Kula, Arie Van Deursen, and Georgios Gousios. Context-aware automated sprint plan generation for agile software development. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 1745–1756, 2024. Open Access version: https://pure.tudelft.nl/ws/portalfiles/portal/233922943/3691620.3695540.pdf.

[133] Barry W Boehm and Philip N. Papaccio. Understanding and controlling software costs. *IEEE transactions on software engineering*, 14(10):1462–1477, 1988.

[134] Adam Trendowicz, Jürgen Münch, and Ross Jeffery. State of the practice in software effort estimation: a survey and literature review. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 232–245. Springer, 2008.

[135] Barbara A Kitchenham and Shari Lawrence Pfleeger. Principles of survey research: parts 1 — 6. *ACM SIGSOFT Software Engineering Notes*, 26-28, 2001 - 2003.

[136] Mark Kasunic. Designing an effective survey. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2005.

[137] Magne Jørgensen and Kjetil Molokken-Ostvold. Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12):993–1007, 2004.

[138] Fritz Strack. "order effects" in survey research: Activation and information functions of preceding questions. In *Context effects in social and psychological research*, pages 23–34. Springer, 1992.

[139] Jefferson Seide Molléri, Kai Petersen, and Emilia Mendes. Survey guidelines in software engineering: An annotated review. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–6, 2016.

[140] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[141] Pearl Judea. Causality: models, reasoning, and inference. *Cambridge University Press. ISBN 0*, 521(77362):8, 2000.

[142] SD Conte, HE Dunsmore, and VY Shen. Software engineering metrics and models. 1986, redwood city.

[143] Tron Foss, Erik Stensrud, Barbara Kitchenham, and Ingunn Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE transactions on software engineering*, 29(11):985–995, 2003.

[144] Barbara A Kitchenham, Lesley M Pickard, Stephen G. MacDonell, and Martin J. Shepperd. What accuracy statistics really measure. *IEE Proceedings-Software*, 148(3):81–85, 2001.

[145] Dan Port and Marcel Korte. Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 51–60, 2008.

[146] Y Miyazaki, M Terakado, K Ozaki, and H Nozaki. Robust regression for developing software estimation models. *Journal of Systems and Software*, 27(1):3–16, 1994.

[147] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[148] John Noll and Sarah Beecham. Measuring global distance: A survey of distance factors and interventions. In *International Conference on Software Process Improvement and Capability Determination*, pages 227–240. Springer, 2016.

[149] Deborah Hartmann and Robin Dymond. Appropriate agile measurement: using metrics and diagnostics to deliver business value. In *AGILE 2006 (AGILE'06)*, pages 6–pp. IEEE, 2006.

[150] Stuart E Dreyfus and Hubert L Dreyfus. A five-stage model of the mental activities involved in directed skill acquisition. Technical report, California Univ Berkeley Operations Research Center, 1980.

[151] Caroline Aube and Vincent Rousseau. Team goal commitment and team effectiveness: the role of task interdependence and supportive behaviors. *Group Dynamics: Theory, Research, and Practice*, 9(3):189, 2005.

[152] Katherine M Chudoba, Eleanor Wynn, Mei Lu, and Mary B Watson-Manheim. How virtual are we? measuring virtuality and understanding its impact in a global organization. *Information systems journal*, 15(4):279–306, 2005.

[153] Ofer Morgenshtern, Tzvi Raz, and Dov Dvir. Factors affecting duration and effort estimation errors in software development projects. *Information and Software Technology*, 49(8):827–837, 2007.

[154] Robert S Huckman, Bradley R Staats, and David M Upton. Team familiarity, role experience, and performance: Evidence from indian software services. *Management science*, 55(1):85–100, 2009.

[155] Kim Dikert, Maria Paasivaara, and Casper Lassenius. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 119:87–108, 2016.

[156] Haiyan Huang and Eileen M Trauth. Cultural influences and globally distributed information systems development: experiences from chinese it professionals. In *Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research: The global information technology workforce*, pages 36–45, 2007.

[157] Mira Kajko-Mattsson. Problems in agile trenches. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 111–119, 2008.

[158] Jyrki Kontio, Magnus Hoglund, Jan Ryden, and Pekka Abrahamsson. Managing commitments and risks: challenges in distributed agile development. In *Proceedings. 26th International Conference on Software Engineering*, pages 732–733. IEEE, 2004.

[159] Nelson Sekitoleko, Felix Evbota, Eric Knauss, Anna Sandberg, Michel Chaudron, and Helena Holmström Olsson. Technical dependency challenges in large-scale agile software development. In *International Conference on Agile Software Development*, pages 46–61. Springer, 2014.

[160] Diane E Strode, Sid L Huff, Beverley Hope, and Sebastian Link. Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6):1222–1238, 2012.

[161] Amany Elbanna and Suprateek Sarker. The risks of agile software development: Learning from adopters. *IEEE Software*, 33(5):72–79, 2015.

[162] Albert L Lederer and Jayesh Prasad. Causes of inaccurate software development cost estimates. *Journal of systems and software*, 31(2):125–134, 1995.

[163] Stein Grimstad, Magne Jørgensen, and Kjetil Molokken-Ostvold. The clients' impact on effort estimation accuracy in software development projects. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10–pp. IEEE, 2005.

[164] Kristian Marius Furulund and Kjetil Molkken-stvold. Increasing software effort estimation accuracy using experience data, estimation models and checklists. In *Seventh International Conference on Quality Software (QSIC 2007)*, pages 342–347. IEEE, 2007.

[165] Rajeev Gupta, K Hima Prasad, and Mukesh Mohania. Automating itsm incident management process. In *2008 International Conference on Autonomic Computing*, pages 141–150. IEEE, 2008.

[166] Brian Fitzgerald, Klaas-Jan Stol, Ryan O'Sullivan, and Donal O'Brien. Scaling agile methods to regulated environments: An industry case study. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 863–872. IEEE, 2013.

[167] Fabiola Moyon, Kristian Beckers, Sebastian Klepper, Philipp Lachberger, and Bernd Bruegge. Towards continuous security compliance in agile software development at scale. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 31–34. IEEE, 2018.

[168] Lotfi ben Othmane, Pelin Angin, Harold Weffers, and Bharat Bhargava. Extending the agile development process to develop acceptably secure software. *IEEE Transactions on dependable and secure computing*, 11(6):497–509, 2014.

[169] Emanuel Giger, Martin Pinzger, and Harald Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56, 2010.

[170] Florence S Downs and J Fawcett. *The relationship of theory and research.* London: McGraw-Hill/Appleton & Lange, 1986.

[171] Jefferson Seide Molléri, Kai Petersen, and Emilia Mendes. An empirically evaluated checklist for surveys in software engineering. *Information and Software Technology*, 119:106240, 2020.

[172] Robert Donmoyer. Generalizability and the single-case study. *Case study method: Key issues, key texts*, pages 45–68, 2000.

[173] Edith Desiree De Leeuw. *Data quality in mail, telephone and face to face surveys.* ERIC, 1992.

[174] Robert M Groves, Robert B Cialdini, and Mick P Couper. Understanding the decision to participate in a survey. *Public opinion quarterly*, 56(4):475–495, 1992.

[175] Adrian Furnham. Response bias, social desirability and dissimulation. *Personality and individual differences*, 7(3):385–400, 1986.

[176] Paul Ralph and Ewan Tempero. Construct validity in software engineering research and software metrics. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 13–23, 2018.

[177] Kjetil Molokken and Magne Jørgensen. A review of software surveys on software effort estimation. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 223–230. IEEE, 2003.

[178] Francesco Corman and Pavle Kecman. Stochastic prediction of train delays in real-time using bayesian networks. *Transportation Research Part C: Emerging Technologies*, 95:599–615, 2018.

[179] Bernd Oreschko, Thomas Kunze, Michael Schultz, Hartmut Fricke, Vivek Kumar, and Lance Sherry. Turnaround prediction with stochastic process times and airport specific delay pattern. In *International Conference on Research in Airport Transportation (ICRAT), Berkeley*, 2012.

[180] Yushan Jiang, Yongxin Liu, Dahai Liu, and Houbing Song. Applying machine learning to aviation big data for flight delay prediction. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 665–672. IEEE, 2020.

[181] Carolyn Mair, Gada Kadoda, Martin Lefley, Keith Phalp, Chris Schofield, Martin Shepperd, and Steve Webster. An investigation of machine learning based prediction systems. *Journal of systems and software*, 53(1):23–29, 2000.

[182] Morakot Choetkiertikul, Hoa Khanh Dam, Truyen Tran, Aditya Ghose, and John Grundy. Predicting delivery capability in iterative software development. *IEEE Transactions on Software Engineering*, 44(6):551–573, 2017.

[183] Morakot Choetkiertikul, Hoa Khanh Dam, and Aditya Ghose. Threshold-based prediction of schedule overrun in software projects. In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, pages 81–85, 2015.

[184] Carlo A Furia, Robert Feldt, and Richard Torkar. Bayesian data analysis in empirical software engineering research. *IEEE Transactions on Software Engineering*, 47(9):1786–1810, 2019.

[185] Carlo A Furia, Richard Torkar, and Robert Feldt. Applying bayesian analysis guidelines to empirical software engineering data: The case of programming languages and code quality. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3):1–38, 2022.

[186] Richard Torkar, Carlo A Furia, Robert Feldt, Francisco Gomes de Oliveira Neto, Lucas Gren, Per Lenberg, and Neil A Ernst. A method to assess and argue for practical significance in software engineering. *IEEE Transactions on Software Engineering*, 48(6):2053–2065, 2021.

[187] Richard McElreath. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2020.

[188] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[189] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering–a decade review. *Information systems*, 53:16–38, 2015.

[190] John A Hartigan, Manchek A Wong, et al. A k-means clustering algorithm. *Applied statistics*, 28(1):100–108, 1979.

[191] Andrea Arcuri and Lionel Briand. A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[192] Silvia Ferrari and Francisco Cribari-Neto. Beta regression for modelling rates and proportions. *Journal of applied statistics*, 31(7):799–815, 2004.

[193] Raydonal Ospina and Silvia LP Ferrari. A general class of zero-or-one inflated beta regression models. *Computational Statistics & Data Analysis*, 56(6):1609–1623, 2012.

[194] Nathan P Lemoine. Moving beyond noninformative priors: why and how to choose weakly informative priors in bayesian analyses. *Oikos*, 128(7):912–928, 2019.

[195] Andrew Gelman. Prior distributions for variance parameters in hierarchical models (comment on article by browne and draper). *Bayesian analysis*, 1(3):515–534, 2006.

[196] Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[197] Aki Vehtari, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. Rank-normalization, folding, and localization: an improved r for assessing convergence of mcmc (with discussion). *Bayesian analysis*, 16(2):667–718, 2021.

[198] Paul-Christian Bürkner, Jonah Gabry, and Aki Vehtari. Approximate leave-future-out cross-validation for bayesian time series models. *Journal of Statistical Computation and Simulation*, 90(14):2499–2523, 2020.

[199] Aki Vehtari, Andrew Gelman, and Jonah Gabry. Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and computing*, 27(5):1413–1432, 2017.

[200] Magne Jørgensen, Morten Welde, and Torleif Halkjelsvik. Evaluation of probabilistic project cost estimates. *IEEE Transactions on Engineering Management*, 2021.

[201] Magne Jørgensen. Evaluating probabilistic software development effort estimates: Maximizing informativeness subject to calibration. *Information and software Technology*, 115:93–96, 2019.

[202] William B Langdon, Javier Dolado, Federica Sarro, and Mark Harman. Exact mean absolute error of baseline predictor, marp0. *Information and Software Technology*, 73:16–18, 2016.

[203] Jakob Runge, Peer Nowack, Marlene Kretschmer, Seth Flaxman, and Dino Sejdinovic. Detecting and quantifying causal associations in large nonlinear time series datasets. *Science advances*, 5(11):eaau4996, 2019.

[204] Noureddine Kerzazi and Foutse Khomh. Factors impacting rapid releases: an industrial case study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, 2014.

[205] Elvan Kula, Ayushi Rastogi, Hennie Huijgens, Arie van Deursen, and Georgios Gousios. Releasing fast and slow: an exploratory case study at ing. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 785–795, 2019. Open Access version: https://pure.tudelft.nl/ws/portalfiles/portal/54964259/Releasing_Fast_and_Slow.pdf.

[206] Maëlick Claes, Mika V Mäntylä, Miikka Kuutila, and Bram Adams. Do programmers work at night or during the weekend? In *Proceedings of the 40th International Conference on Software Engineering*, pages 705–715, 2018.

[207] Ahmed Al-Emran, Dietmar Pfahl, and Günther Ruhe. Dynarep: A discrete event simulation model for re-planning of software releases. In *Software Process Dynamics and Agility: International Conference on Software Process, ICSP 2007, Minneapolis, MN, USA, May 19-20, 2007. Proceedings*, pages 246–258. Springer, 2007.

[208] David Ameller, Carles Farré, Xavier Franch, Danilo Valerio, and Antonino Cassarino. Towards continuous software release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 402–406. IEEE, 2017.

[209] VersionOne. 14th State of Agile Survey. https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494, .

[210] Helen Sharp and Hugh Robinson. Three 'c's of agile practice: collaboration, co-ordination and communication. In *Agile software development*, pages 61–85. Springer, 2010.

[211] Subhas Chandra Misra, Vinod Kumar, and Uma Kumar. Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software*, 82(11):1869–1890, 2009.

[212] Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams, and Marvin Zelkowitz. Empirical findings in agile methods. In *Conference on extreme programming and agile methods*, pages 197–207. Springer, 2002.

[213] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Overcoming barriers to self-management in software teams. *IEEE software*, 26(6):20–26, 2009.

[214] Terry L Dickinson and Robert M McIntyre. A conceptual framework for teamwork measurement. *Team performance assessment and measurement*, pages 19–43, 1997.

[215] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[216] ServiceNow. Workflows for the Modern Enterprise. 2021. https://servicenow.com.

[217] Wayne W Daniel et al. Applied nonparametric statistics. 1990.

[218] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[219] Barry Boehm and Richard Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.

[220] Claudia de O Melo, Daniela S Cruzes, Fabio Kon, and Reidar Conradi. Interpretative case studies on agile team productivity and management. *Information and Software Technology*, 55(2):412–427, 2013.

[221] Vikash Lalsing, Somveer Kishnah, and Sameerchand Pudaruth. People factors in agile software development and project management. *International Journal of Software Engineering & Applications*, 3(1):117, 2012.

[222] Rashmi Popli and Naresh Chauhan. Agile estimation using people and project related factors. In *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 564–569. IEEE, 2014.

[223] Siva Dorairaj, James Noble, and Petra Malik. Understanding team dynamics in distributed agile software development. In *International conference on agile software development*, pages 47–61. Springer, 2012.

[224] J Alberto Espinosa, Sandra A Slaughter, Robert E Kraut, and James D Herbsleb. Familiarity, complexity, and team performance in geographically distributed software development. *Organization science*, 18(4):613–630, 2007.

[225] Thomas Tan, Qi Li, Barry Boehm, Ye Yang, Mei He, and Ramin Moazeni. Productivity trends in incremental and iterative software development. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 1–10. IEEE, 2009.

[226] Katrina D Maxwell and Pekka Forselius. Benchmarking software development productivity. *Ieee Software*, 17(1):80–88, 2000.

[227] Robert W Zmud. Management of large software development efforts. *MIS quarterly*, pages 45–55, 1980.

[228] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[229] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[230] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[231] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[232] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[233] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[234] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[235] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2):103–130, 1997.

[236] Jin Huang and Charles X Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering*, 17(3):299–310, 2005.

[237] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. Variable selection using random forests. *Pattern recognition letters*, 31(14):2225–2236, 2010.

[238] Fabian Kortum, Jil Klünder, and Kurt Schneider. Behavior-driven dynamics in agile development: The effect of fast feedback on teams. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 34–43. IEEE, 2019.

[239] Victor R Basili and Robert W Reiter Jr. An investigation of human factors in software development. *IEEE Computer*, 12(12):21–38, 1979.

[240] Fabian Kortum, Jil Klünder, and Kurt Schneider. Don't underestimate the human factors! exploring team communication effects. In *International conference on product-focused software process improvement*, pages 457–469. Springer, 2017.

[241] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd international conference on software engineering*, pages 1–10, 2011.

[242] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[243] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9-10):833–859, 2008.

[244] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development, 2012.

[245] Rashina Hoda, Norsaremah Salleh, and John Grundy. The rise and evolution of agile software development. *IEEE software*, 35(5):58–63, 2018.

[246] Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. Multi-sprint planning and smooth replanning: An optimization model. *Journal of systems and software*, 86(9):2357–2370, 2013.

[247] Hoa Khanh Dam, Truyen Tran, John Grundy, Aditya Ghose, and Yasutaka Kamei. Towards effective ai-powered agile project management. In *2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER)*, pages 41–44. IEEE, 2019.

[248] Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. Sprint planning optimization in agile data warehouse design. In *Data Warehousing and Knowledge Discovery: 14th International Conference, DaWaK 2012, Vienna, Austria, September 3-6, 2012. Proceedings 14*, pages 30–41. Springer, 2012.

[249] Marco A Boschetti, Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. A lagrangian heuristic for sprint planning in agile software development. *Computers & Operations Research*, 43:116–128, 2014.

[250] S Jansi and KC Rajeswari. A greedy heuristic approach for sprint planning in agile software development. *International Journal for Trends in Engineering & Technology*, 3(1):18–21, 2015.

[251] Wisam Haitham Abbood Al-Zubaidi, Hoa Khanh Dam, Morakot Choetkiertikul, and Aditya Ghose. Multi-objective iteration planning in agile development. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 484–493. IEEE, 2018.

[252] Nilay Ozcelikkan, Gulfem Tuzkaya, Cigdem Alabas-Uslu, and Bahar Sennaroglu. A multi-objective agile project planning model and a comparative meta-heuristic approach. *Information and Software Technology*, 151:107023, 2022.

[253] R Scott Harris and Mike Cohn. Incorporating learning and expected cost of change in prioritizing features on agile projects. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 175–180. Springer, 2006.

[254] Philip Achimugu, Ali Selamat, Roliana Ibrahim, and Mohd Naz'ri Mahrin. A systematic literature review of software requirements prioritization research. *Information and software technology*, 56(6):568–585, 2014.

[255] Rami Hasan AL-Ta'ani and Rozilawati Razali. Prioritizing requirements in agile development: A conceptual framework. *Procedia Technology*, 11:733–739, 2013.

[256] Meghann L Drury-Grogan. Performance on agile teams: Relating iteration objectives and critical decisions to project management success factors. *Information and software technology*, 56(5):506–515, 2014.

[257] Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE software*, 25(1):60–67, 2008.

[258] Charles Spearman. The proof and measurement of association between two things. 1961.

[259] Allysson Allex Araújo, Matheus Paixao, Italo Yeltsin, Altino Dantas, and Jerffeson Souza. An architecture based on interactive optimization and machine learning applied to the next release problem. *Automated Software Engineering*, 24:623–671, 2017.

[260] Tadashi Dohi, Yasuhiko Nishio, and Shunji Osaki. Optimal software release scheduling based on artificial neural networks. *Annals of Software engineering*, 8(1-4):167–185, 1999.

[261] Robert Dorfman, Paul Anthony Samuelson, and Robert M Solow. *Linear programming and economic analysis*. Courier Corporation, 1987.

[262] George B Dantzig. Linear programming. *Operations research*, 50(1):42–47, 2002.

[263] Siw Elisabeth Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10–pp. IEEE, 2005.

[264] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

[265] Thiago do Nascimento Ferreira, Allysson Allex Araújo, Altino Dantas Basílio Neto, and Jerffeson Teixeira de Souza. Incorporating user preferences in ant colony optimization for the next release problem. *Applied Soft Computing*, 49:1283–1296, 2016.

[266] Marc Goerigk and Michael Hartisch. A framework for inherently interpretable optimization models. *European Journal of Operational Research*, 310(3):1312–1324, 2023.

[267] Robert J Lempert, Benjamin P Bryant, and Steven C Bankes. Comparing algorithms for scenario discovery. *RAND, Santa Monica, CA*, 2008.

[268] Dieter Klein and Sören Holm. Integer programming post-optimal analysis with cutting planes. *Management Science*, 25(1):64–72, 1979.

[269] Harvey J Greenberg. The use of the optimal partition in a linear programming solution for postoptimal analysis. *Operations Research Letters*, 15(4):179–185, 1994.

[270] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert systems with applications*, 41(4):2065–2073, 2014.

[271] Alistair G Sutcliffe, Neil AM Maiden, Shailey Minocha, and Darrel Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on software engineering*, 24(12):1072–1088, 1998.

[272] Björn Regnell, Per Runeson, and Thomas Thelin. Are the perspectives really different?–further experimentation on scenario-based reading of requirements. *Empirical Software Engineering*, 5:331–356, 2000.

Curriculum Vitæ

Elvan Kula

1993/08/27	Date of birth in Winschoten, The Netherlands

Education

6/2019–5/2024	Ph.D. Student, Software Engineering Research Group, Delft University of Technology, The Netherlands, *Modeling Effort Estimation and Planning in Large-Scale Agile Software Development*, Promotor: Prof. Dr. Arie van Deursen Supervisor: Dr. Georgios Gousios
9/2017–5/2019	M.Sc. Computer Science, Delft University of Technology, The Netherlands, Thesis: *Releasing Fast and Slow: Characterizing Rapid Releases in a Large Software-Driven Organization*
9/2013–7/2016	B.Sc. Computer Science, Delft University of Technology, The Netherlands, Thesis: *AeroVision: An Integrated Solution to the Optimization and Visualization of Aircraft Noise*

Work Experience

9/2023–Present	Strategic Data Analyst Royal Schiphol Group, Amsterdam, The Netherlands
6/2019–8/2023	Chapter Lead in Data Science and AI Research ING Bank, Amsterdam, The Netherlands

2/2018–5/2019	Graduate Intern ING Bank, Amsterdam, The Netherlands
8/2016–2/2018	Software Engineer in Operations Research & Decision Support KLM Royal Dutch Airlines, Amstelveen, The Netherlands
8/2014–1/2016	Student Mentor and Teaching Assistant Delft University of Technology, Delft, The Netherlands

List of Publications

1. *Elvan Kula*, Ayushi Rastogi, Hennie Huijgens, Arie van Deursen, and Georgios Gousios: Releasing fast and slow: an exploratory case study at ING. Published in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2019.

2. *Elvan Kula*, Eric Greuter, Arie van Deursen, and Georgios Gousios: Factors affecting on-time delivery in large-scale agile software development. Published in IEEE Transactions on Software Engineering (TSE), 2021.

3. *Elvan Kula*, Arie van Deursen, and Georgios Gousios: Modeling team dynamics for the characterization and prediction of delays in user stories. Published in Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2021.

4. *Elvan Kula*, Eric Greuter, Arie van Deursen, and Georgios Gousios: Dynamic Prediction of Delays in Software Projects using Delay Patterns and Bayesian Modeling. Published in Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2023.

5. *Elvan Kula*, Arie van Deursen, and Georgios Gousios: Context-Aware Automated Sprint Plan Generation for Agile Software Development. Published in Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2024.

Included in this thesis

Won a Distinguished Paper Award