Story level risk assessment in agile software development using machine-learning

Dries Meerman

driesmeerman@student.uva.nl

May 18, 2022 51 pages

Academic supervisor: Zhiming Zhao, z.zhao@uva.nl
Daily supervisor: Guus Hutschemaekers, guus.hutschemaekers@plat4mation.com
Host organisation: App4mation, https://app4mation.com

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Project failure is a common occurrence in the IT-industry. Risk management is an important part of preventing software project failure. Agile development methods such as Scrum try to prevent risk by being flexible and having shorter feedback cycles. This allows development teams to quickly change direction as needed. During these shorter feedback cycles a large amount of contextual data around the product is generated. This is captured in the product and sprint backlogs, once sprints are completed this data stays available.
Research found that risk management should be continuously applied in a feedback loop to lead to better outcomes. This is why we propose an automated way of doing risk assessment using machine-learning. By using automation it can be integrated into a feedback loop without requiring costly human labour. We use engineering logs generated during development, to train machine-learning models using the user stories outcomes as labels. Data used for machine-learning training purposes requires labels whereas historical project data is often unlabeled. We tried to relate user stories to risk by finding features of a story that could be used as risk labels. For this purpose we researched risk and surveyed industry experts to rank risks.
Among Scrum practitioners the development process is not identical, the available engineering logs will also vary. Our system has to be able to process data coming from various sources, to do this we integrated a configuration component. This allows the system to be able to use information logged in various Issue Tracking Systems (ITS). Before this historical data can be passed to the machine-learning algorithms the information needs to be transformed into a suitable format. A large part of this data is textual and requires Natural Language Processing (NLP). We compare bag of words (BoW) method with a two phased approach of passing text into existing machine-learning models which are then used as input points for the final predictive model.
These models, trained on historical data can be used to assess the risk in newly generated user stories. Because this is an automated process it can be continuously applied and the resulting prediction can be attached to the existing Scrum process.
To improve the performance of the prediction we deployed machine learning ensemble algorithms, which combine multiple underlying models. We found that our best case machine learning ensemble (Random Forrest) performed 26% better than human experts on the same risk assessment.

Contents

Chapter 1

Introduction

Project failure [1–3] and budget overruns are common in the the IT industry [4, 5]. Events that could cause these unwanted outcomes are called risks; detecting them (i.e., risk assessment) in an early phase and controlling them effectively (i.e., risk control) is crucial for the success of a project [6]. However, analyzing and detecting risks often relies on the experience of human experts, such as project managers, yet it remains one of the least practiced project management knowledge areas [7].
Over time software development methods have evolved, early processes followed sequential phases like a waterfall. Going from phase to next until the end of the project is reached, attempting not to go backwards in phases. This structure enforced the project members to weigh technical decisions, since changing a plan while building is harder than while designing. Having such a more fixed design at the start makes it hard to anticipate on customer needs which might change. Or might not have been adequately understood by the designers. As a response to this "Agile" approaches started to become popular. Instead of a waterfall with a big bang delivery at the end. The project is split into short iteration cycles with a constant feedback loop. Not a big design that is hard to change, but a design of what is going to be created shortly. If that takes the product in the wrong direction, then new designs can be made and the direction can be altered.
In traditional waterfall approaches, risk management often takes the form of risk management techniques with different explicit phases. Risk identification, risk analysis, risk monitoring and risk response. Since there is less flexibility to handle risks it is more important to prevent them upfront. In agile processes where there might be more unknowns due to the rapidly changing environment, there is an orientation towards flexibility [8]. Due to the shorter cycles in agile processes, the amount of overhead some risk management activities take can appear to be not worthwhile.
Agile development methodologies have demonstrated their advantages over traditional methods in reducing time to market, improving productivity, and having a closer alignment with business needs[9]. A widely used agile process is Scrum [10] this is the process has our focus due to its popularity and availability of data. In Scrum key stakeholders in the software life cycle, e.g., product owners and software developers, are closely engaged in the iterative processes. The product owner generates a development plan (i.e., product backlog), based on user requirements and feedback, discusses with the developers to plan specific development activities (i.e., sprint backlogs), and schedules them as iterative small cycles (i.e., sprints). A Scrum master closely monitors the progress and assures transparency to stakeholders [11]. Agile processes can be supported using different online tool systems, which are often operated as software as services, e.g., Jira[1] and Azure DevOps[2]. However, not all teams use technology to manage these processes.
Compared to classical waterfall methods, agile methodologies show clear advantages in reducing development risks of project delay due to its iterative process management and a short cycle of software delivery. However, other types of risks, e.g., unclear user story, inefficient decomposition, and scheduling of the sprints backlogs, remain a big challenge[6, 12].
Detecting risks in the agile processes can reduce unnecessary iterations and avoid the failure of sprints. Many of such risks can be traced back to the quality of the user stories and requirements. Poor quality user stories can result in misinterpretation in the process of sprint backlog generation and planning. However, at the moment, most of the risk detection on the user stories still rely on human experts, and many risks cannot be detected in time.
There is a large extent of contextual information about projects captured in the backlogs of products, sprints and logs related to the code's version history. That historical information provides valuable sources for analyzing and characterizing risks. Effective detecting development risk from historical data, particularly from user stories, emerges as an important need in agile development.
A common practice used along side Scrum is DevOps, or rather it is a set of practices, philosophies and tooling. It aims to shorten the development life-cycle by using automation and closer integration between development and operation teams.
Now lets take a look at the types of logs that can be generated by the combination of Scrum and Devops.

[1] https://www.atlassian.com/software/jira
[2] https://azure.microsoft.com/en-us/services/devops/

Followed by a more detailed look at development cycle in scrum including DevOps tooling, which is commonly used to reduce risk and manual effort by using automation. We will start with figure 1.1 which shows the Scrum cycle and the development cycle[3]. Scrum starts with a product backlog which usually consists of system requirements in the form of user stories[4]. Stories are commonly written in the following format "As an <actor> I want to <do action> so I can <achieve goal>". They are created by the product owner based on the direction they want to take the product in and feedback on the existing product.

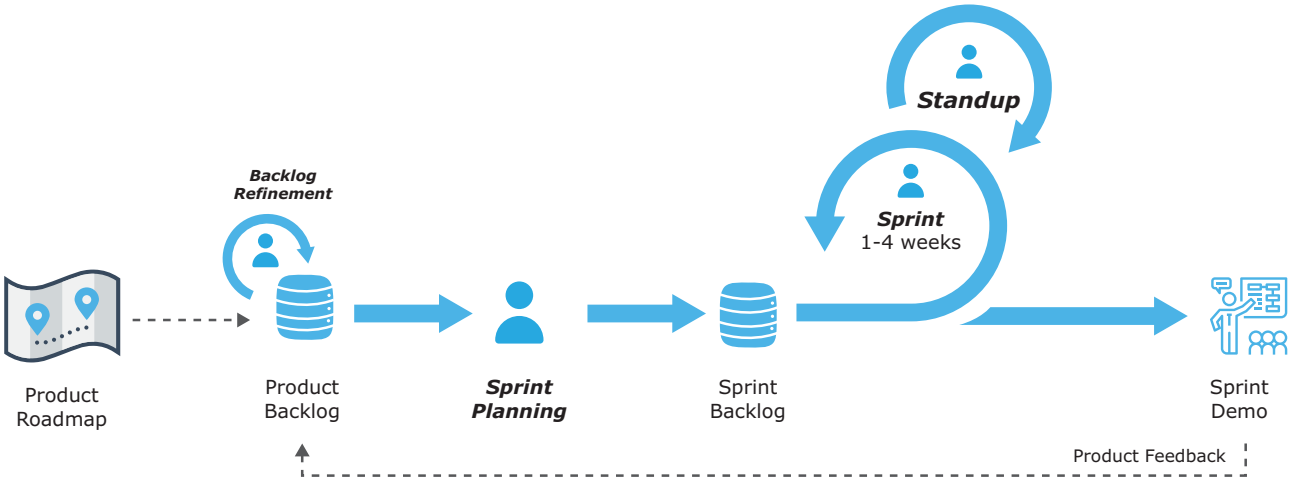

Figure 1.1: Scrum development cycle.

Stories can contain related data: estimated effort in the form of story points, assigned user who is working on the task, related Scrum epic and theme and priority, classification (e.g type defect, feature, etc).
In the following phase, sprint planning the development teams chooses work from the backlog and turn it into plan for the coming increment. This increment should be a usable version of the project and is delivered at the end of the sprint. The sprint takes a fixed amount of time usually between 1-4 weeks, this is decided by the team itself.

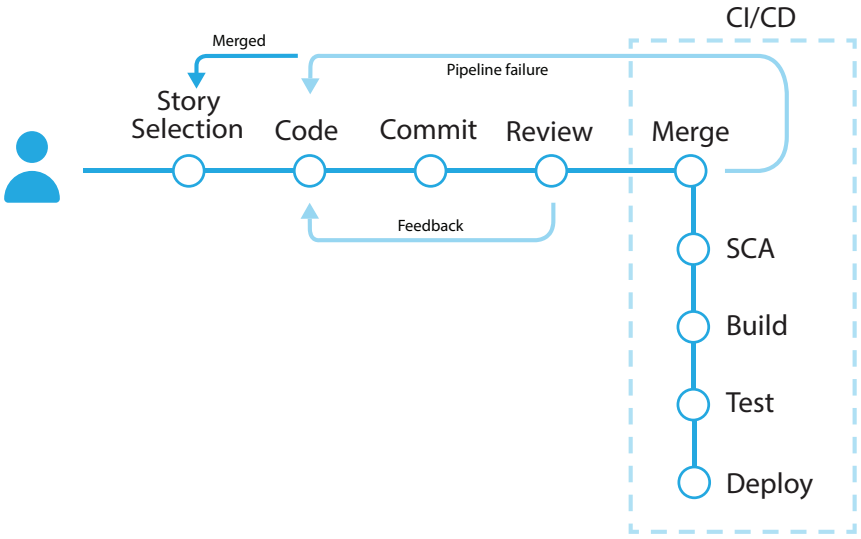

Figure 1.2: Sprint development cycle.

During the sprint developers create code for their stories, which is committed to a version control system (VCS)[5], after the story is finished the code is reviewed. This can lead to a few iteration cycles of code changes due to feedback, after which the code will be accepted and merged into the existing product. A merge often triggers a DevOps pipeline, where the code will be analyzed, build and tested. If these steps succeed the code can be deployed or the code might need to be modified some more. Depending on the maturity of the DevOps process in a company the level of automation of these steps can differ.

[3]Activities are highlighted in bold
[4]This is not strictly required by the Scrum Guide [13]
[5]Code committed to VCS is often linked to the requirement to comply with ISO9001.

In this research we use machine-learning techniques to create a predictive model for risk assessment on user story level and related engineering logs. To the best of our knowledge machine learning based risk detection on user story level has not yet been studied. In Scrum it is risk is minimized by short iteration cycles allowing for fast responses and by minimizing the amount of work that will be impacted by problems. Our goal is to explore the usage of data captured in agile processes, to shorten the risk detection cycle. By using automation and lowering the required manual effort from human experts, we hope to more deeply integrate risk assessment into the agile process. We hope this leads to more successful sprints.

Our research answers the following research questions:

RQ1: What kind of pre-processing is required on engineering logs for usage in risk predicting machine-learning models?

RQ2: What kind of risk labels can be found in user stories?

RQ3: How can machine-learning-based automation improve existing risk prevention in agile development processes?

RQ3.1: How do different machine-learning algorithms behave when using user story data as input?

RQ4: What can be improved by applying ensemble machine-learning algorithms?

Our research makes the following contributions:

- a rated list of risks
- a configurable machine-learning model training pipeline
- insight into the performance of different kind of models compared to a human baseline

Thesis outline
The rest of the thesis is structured as follows, we start with the current state of the art surrounding related software methodologies, risk and machine learning. This is followed by research into risk and our survey to rank important risks which will later be used in the experiments. Following the risk chapter we explain the system design, first the architecture followed by the implementation. Then we evaluate the system by performing an experiment. We then try to improve the results of this experiment using ensemble algorithms, the changes are then evaluated with more experiments. After this we discuss the research questions and threats to validity based on the results of the experiments. We then end the document with our conclusion and a future work chapter.

Chapter 2

State of the Art

This chapter is dedicated to existing research from relevant domains. We do this to relate our research to existing literature and to retrieve knowledge on our research questions.
This research cuts into multiple domains therefore this chapter will be split up into three main parts:

- Software methodologies
- Risk
- Machine Learning

In the software methodologies domain we want to find out more information about the processes we are trying to improve. In the risk domain we want to learn about the types of risk in software development. We need to be able to identify risks to be able to know whether our system is able to identify risks. What kind of automation previously has been used in the risk management space and how this interacted with the software development process. Because tools are only use full if they can be implemented. Our approach wants to use machine learning in assessing risk, but how can ML be implemented in our system? To find a relevant ML algorithm to use we need to know which algorithms are suitable for which applications. We hope to find research which is trying to achieve similar results with different data, or research which uses similar data for other goals. This should help us in designing and building our risk assessment solution. At the end of this chapter there will be a summary about the gaps in the existing literature that this thesis is researching.

2.1 Method

The main entry point for finding research will be the Google Scholar search engine. We use a combination of research from all time, from 2017 and upward to find recent literature. We use search terms containing the following keywords for the risk and requirements engineering domains:
"important, risks, software development, software project, risk evaluation, risk detection, agile, requirements, risk knowledge". We chose to intertwine our search, as our focus lies with project / requirements related risks. Masso et al [14] studied the literature around risk in software development also containing solutions to common problems. This study was a great entry point into the field of risk management. The terms we used for machine learning where the following:
"machine learning on user stories, machine learning literature survey for software risks, machine learning algorithms literature survey, when to use unsupervised learning, reinforcement learning, supervised learning, machine learning models". For the last domain DevOps we these terms:
"devops, devops risk management, implementing new tools in devops process". From this retrieved corpus of literature, we look at the related work sections and relevant works in literature studies.

2.2 Software methodologies

Nobody wants their projects to fail, throughout the history of software engineering many methods have been researched and tried to improve project outcomes. Yet many projects still get delivered past their deadlines over budget or not satisfying the users [1–3]. Processes around software development have been created to prevent these types of failures. This section explains some background around these processes and how they try to prevent risks from causing failure.

SDLC (Systems Development Life Cycle) is a concept which describes the phases of a software / hardware development project. Consisting of roughly five phases: "Planning, Analysis, Design, Implementation and Maintenance". In a 1970 paper Royce described his take on the waterfall model [15] which is an implementation of the SDLC. Royce found that projects following this model had a higher likely hood of succeeding than projects which did not. The waterfall model consists of five main steps:

1. Program design (System and software requirements)

2. Document the design (Detailed description of the software components)
3. Do it twice (The software the customer receives should be a second iteration)
4. Plan, control and monitor testing (Testing should be done thoroughly before delivery)
5. Involve the customer (Final validation before starting operations)

With a focus on documenting each step well, "During the early phase of software development the documentation is the specification and is the design." [15]. This process tries to prevent *risk* by analysing the requirements up front, iterating on the design. Followed by prototyping the software rather than only testing and delivering a single version. So that problems are prevented before the testing phase is reached, if problems are found during testing they are hopefully superficial. Finally a structured phase where the customer is consulted as requirements are often interpreted differently by different parties. It is important to get commitment from the customer even after previous agreement, as requirements are sometimes reinterpreted in different ways. Doing this during all phases of the projects can lead to chaotic changes in requirement. That is why this model recommends doing it at three points: after the system requirement gathering, after the preliminary program design and after testing before going into production. Since then many methodologies have appeared to try and increase the success rate of software projects, many using ideas originating from this model.

Agile has been a popular way of the developing software the past few decades [16], it is based on the same core concepts of SDLC. But it has a few key differences, rather than going in one direction like a waterfall. Agile methods focus on a more iterative process, where the same steps are repeated in a cycles. Instead of having one large test phase at the end, testing happens during every cycle. Instead of heavy documentation which is likely to get outdated, focus on delivering working code each cycle. These methodologies assume that the requirements won't be perfect and have to be revised. These types of revisions take time, more documentation would lead to more time needed to keep everything up to date. The focus is working software, which is continuously being demoed to stakeholders which can give feedback. This solution tries to prevent risk by being flexible and being able to adopt to change well. There are many forms of agile software development, our focus will lie with Scrum. As it is currently the most popular form of agile [10, 11, 17, 18]. This was also found by Tavares et al, because of its popularity they performed a study comparing literature around risk in Scrum to a survey by practitioners. Risk management in projects using Scrum and even software projects in general seems lacking. They also research the effect of agile methodologies not explicitly suggesting risk management processes. The research identifies a set of practices that practitioners and literature see as possibly beneficial for risk management. The most important finding was that risk management must be applied continuously in a feedback loop [19]. Some research has been done using ML to estimate story points (effort) of new work [20, 21]. A study by Satapathy and Rath compares the effectiveness of different ML techniques for this purpose [22].

DevOps the word is a portmanteau combining the words development and operations. In a DevOps culture these teams are joined these separate teams are joined, to reduce the gap between operations and development leading to faster responses. It is a layer of automation on top of SDLC phases, the build (artifact creation), test, release and monitor phases. With DevOps tools are used to increase speed, reliability and scale. By giving teams full responsibility for specific services, they can release smaller chunks more often. To be able to handle this speed increase without introducing errors, software tool-chains are used. It has been following an upward trend for the last few years see figure 2.1. In no small part due to DevOps being a solution to faster reliable releases, which is something the industry is looking for.

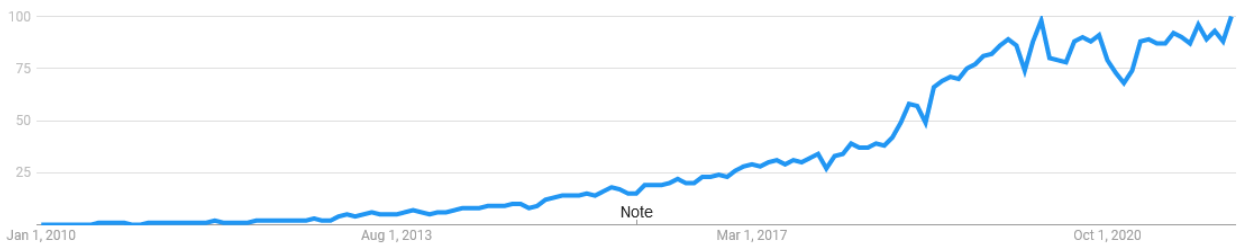

Figure 2.1: Google Trends "devops" 2010/01 - 2022/04 [23]

DevOps tries to help projects succeed by increasing the speed of the feedback loop. Using automated tools to ensure "correctness" of the deliverable. This makes DevOps well suited for aiding risk management. There is overlap in the DevOps philosophy and the findings by Tavarez et al, where they saw benefit in risk management activities being executed in a feedback cycle. The common DevOps cycle is shown in figure 2.2.

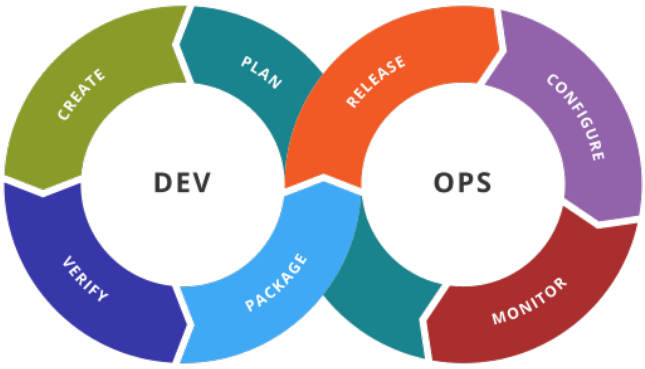

Figure 2.2: DevOps life-cycle [24]

To know how we can integrate a tool in a DevOps process, we need to know how this looks like in practice. By executing an interview based study on 6 organizations around DevOps practices. Erich et al first describe literature surrounding DevOps followed by observations around these organizations. They observed that most organizations had a positive experience with DevOps and only minor problems where encountered [25].
How has DevOps helped with risk management? To answer this we look at a study by Laukkarinen et al, they research DevOps in regulated environments [26]. In these environments strict risk management is often required. They found that there is a gap between existing tooling and regulations around these types of projects. Mainly in the form of the precision in which requirements are traced to delivered code.

2.3 Risk

We need a better understanding of risk to channel our research into a specific direction as risk itself is a broad topic. We will start with risk definitions so we can compare risks. Then look into risks themselves followed by ways of automating risk management.
Karadesh et all define it as "an event that might cause obstruction to projects progress" [27]. It is important to not look at all risks equally, as different risks can lead to very different problems. We want to focus on risks that are more likely to cause problems to find those we need ways of ranking them. This can be done by looking at probability and impact. For this we have used the impact and probability definitions from Li et al [28]. By looking at the impact and probability of a risk a rank, low, medium or high can be identified.

Impact	Outcome	Numerical
Critical	Project will fail.	5
Serious	Major cost / schedule increases.	4
Moderate	Some cost / schedule increases, MVP will not be met.	3
Minor	Small cost / schedule increase, only the MVP will be met; however, some secondary requirements will not be met.	2
Negligible	No impact on project outcome.	1

Table 2.1: Risk impact definition [28]

Probability(%)	Description	Numerical
0-20	Very unlikely the risk will occur	1
21-40	Unlikely the risk will occur	2
41-60	Even likelihood the risk will occur	3
61-85	Likely the risk will occur	4
86-100	Very likely the risk will occur	5

Table 2.2: Risk probability definition [28]

	N (Negligible)	Mi (Minor)	Mo (Moderate)	S (Serious)	C (Critical)
86-100	Low	Medium	High	High	High
61-85	Low	Medium	Medium	High	High
41-60	Low	Low	Medium	Medium	High
21-40	Low	Low	Low	Medium	Medium
0-20	Low	Low	Low	Low	Medium

Table 2.3: Risk ranking [28]

Alves et al did a literature study to find important risks, they validated these risks using a case study of multiple teams of students during a course [29]. They showed that risks identified in studies during the 90's where still relevant. Their case study was based on projects done in an academic environment, so the results might differ in projects in the IT industry. Pasha et al describe the differences between risk analysis in larger and smaller systems, they also propose risk mitigation strategies for larger systems [30]. The paper also contains common risk factors and mitigation strategies for each of the defined risk categories.
What is keeping companies from using risk management? The paper by Odzaly et al researches barriers to risk management. They surveyed a group of 18 experienced project managers and found risk awareness, but low tool usage. Evidence was found that main barriers to performing risk management activities, are related to the perceived cost and value "Risk identification and monitoring, in particular, are perceived as high effort and costly." [31]. They conclude that value of risk management activities should be more visible to show the value.

Risk management automation is not a new concept, to be able to answer research question one we need to know about existing research in this space. From the literature around risk and the software development life-cycle we see that there is a need to make risk more visible and a more integrated part of the agile work flow. While at the same time risk activities aren't practiced as much [19], we hope automation can serve as a solution to both these challenges.
A good example of automated risk management is RisCal. In 2013 Haisjackl et al, created RisCal, a tool which supports "risk identification, analysis and prioritization" [32]. This tool does not automate risk analysis, it has a focus on technical risks and the business implications of those risks. It helps guide decisions based on manual and automated metrics. Monetary impact is given as an manual metric that a person would need to choose the value for. An example of an automated metric that could be supported is cyclomatic complexity, however the related rules on what to do with the values need to be configured. The goal of RisCal is to add structure to the risk management process for arbitrary risks.
Part of the related engineering logs available to us are GIT commits which are linked to user stories. We wanted to know how commits have been used to find risk. Research by Barish et al tries to identify risky code by mining commit messages, using a Naïve Bayes classifier. They use a data driven approach to mark code that is known to be risky based on terms in commit messages. Then use this as a data set to find risky commits even if the same risky terms where not found in commit message [33].
Research by Mahdi et al [34] has been done to predict where defects would occur as to focus the testing and verification efforts. They found that 53% of the defects occurred in 8% of the code, their prediction model had 90% accuracy. Their method uses a random forest approach which was shown to have better results than "Naive Bayes, Logistic Regression and Decision Trees" with their data.
To answer the question how can we automate risk analysis, we found the following paper by Zavvar et al. it classifies risk using Support Vector Machine (SVM) based method [35]. They also compare the performance of risk classification using SVM with other algorithms such as K-means, Self organizing map, Accuracy rate and Area under curve.
Sangaiah et al use fuzzy multi-criteria decision making to automatically aid risk assessment, they validate their approach against a data set of 40 projects showing their approach could be effective [36]. It contains specific risks in the categories of "requirements, estimations, planning, team organization, project management."
A study by Joseph tries identify risk in a different way. By generating relevant risk prompts, instead of trying to classify risks in the project. They use ML to generate risk prompts based on a software projects characteristics [37]. They use Artificial Neural Network (ANN)'s on a list of collected risk prompts, to show relevant ones to a users project. This was compared to risk prompt selection based on keywords which returned around 50% more risk prompts.
In 2018 Odzaly et al wrote a paper where they researched the use of software agents to automate risk management tasks [38]. They demonstrated that the agents are of use for detecting risk, and that risk management in agile context needs to be lightweight like the agile frameworks. The agents where also able to dynamically react to changes in the project environment helping to minimize human effort. Their tool makes use of a risk register

their identification agent then "triggers" risks once a risk in the register matches certain rules in the system. So their approach does require careful examination of risks and deciding of rules upfront to work optimally.
We have compiled a table 2.4 of risk related papers, showing the granularity and whether the paper also contained mitigation's for these risks.

Paper	Year	Risk granularity	Mitigation	Notes
[39]	2015	Category	X	
[40]	2017	General		Risks listed by category.
[35]	2017	General		Risks listed by category.
[41]	2018	General	X	Risks grouped by concern (category).
[38]	2018	General		Defined risk in one list and challenges by category in another.
[30]	2018	General	X	Categorized by project phase, ranked for small and large scale projects. A separate list of mitigation tools by project phase.
[42]	2019	General	X	
[43]	2020	General / Category		Both a list of risks and a list of categories, with rankings comparing Kanban, Scrum and large, small companies.
[44]	2021	General		
[45]	2021	General		Comparison between Disciplined Agile Delivery (DAD) and SAFE on whether risks are addressed.
[29]	2021	General		

Table 2.4: Papers listing risks

2.4 Machine learning

In this section we look at work that has been performed around risk using similar data, or using machine learning techniques.

Research by Hu et al, looked at **software risk prediction** in the context of outsourced software projects with a focus on cost sensitivity [46]. In their approach they used an ensemble-based prediction model, as input for their model they used project level variables with a focus on cost based factors. Examples of these are development cost, requirement stability, collaboration of client team and more; these where scaled as high medium or low in a survey. Tavares et al, developed a tool for risk analysis and management practices for iterative cycles which are common in agile processes [12]. This tool is part of a method which adds risk related activities to agile methods such as creating risk response plans. The tool uses data from the same agile cycle as explained in figure 1.1, but expects more input from people rather than automated data retrieval.
There has been more research into **software effort estimation** using machine learning. For example work by Chou et al[47] created a model to help with personnel planning activities, to achieve this they combined an evolutionary SVM with an fast messy genetic algorithm. They used 17 input variables related to enterprise resource planning and software development effort. The work by Choetkiertikul et al [20] like our research used user stories as a data-source for machine learning algorithms, the difference is that they predict effort (story points) instead of risks. They compared two deep learning algorithms being, long short-term memory (LSTM) and recurrent highway network (RHN). Nassif et al, used a treeboost ensemble to predict software effort in use-case points [48]. Due to the confidentiality of uml diagrams which are usually used to calculate use-case points, they used a questionnaire as input data. They asked questions relating to the number of use-cases in a project, the amount of transactions in the use-cases, the actual effort and factors relating to productivity. Pospieszny et al, also researched effort estimation using an ensemble that contained an SVM, ANN and a generalized linear model [21]. Their research uses 11 project characteristics as input variables, examples of these are industry sector, application type and whether or not the project was agile.
There has also been work into automated **defect prediction**. For example the work by Amasaki et al which focused on simplifying input data, to allow for better cross-project defect predictions [49]. They used static

code analysis metrics as inputs for their models, they where created using the following algorithms: Logistic regression, Random forests, Naïve Bayes and SVM with RBF kernel. Fu and Menzies used supervised and unsupervised learning, with the goal of defect prediction [50]. They compare supervised against unsupervised techniques for defect predictions. They use 14 change related metrics as inputs, examples are, lines of code added, number of modified files and number of developers that changed the modified files. For the supervised algorithms they used, a modified linear regression model, K-NN and Random Forests; they proposed their own algorithm (OneWay) to select the best performing one for a given data set. They used a set of 12 unsupervised algorithms to compare to, based on earlier research by Yang et al [51].
There has also been research in finding problems with the product an example of this is finding technical **debt**. Maldonado et al, used comments in code as input source to detect, design, requirement and technical debt. They used the Stanford Classifier with different underlying classifiers, the maximum entropy classifier, Naive Bayes and Binary classifiers[52].
Finally we have created an overview creating a table looking at algorithms that have been used in existing research. That could help guide us in picking a algorithm see table 2.5.

Paper	**Year**	**Model / Technique**	**Notes**
[46]	2015	SVM, ensemble (bagging)	Contains methodology on how they created their ensemble.
[53]	2014	Naïve Bayes	NB was compared against ANN, NB outperformed. Focus on effort estimation.
[54]	2009	SVM	The SVM outperformed a ANN
[55]	2018	ANN	cost estimation, used cuckoo optimization algorithm
[47]	2012	SVM, FMGA	software effort estimation
[56]	2016	Random Forest	For defect prediction, compared against NB DT
[20]	2016 (arXiv) 2019 (IEEE)	Deep learning (LSTM, RHN)	Effort estimation, using deep learning to get a deeper meaning of the story
[57]	2019	PageRank	Security risk focus not software risk
[49]	2015	K-NN, NB	Defect prediction focus on data simplification
[48]	2012	Treeboost	Model was compared to multiple linear regression model, for estimation

Table 2.5: ML model selection

2.5 Research Gap

The domain of risk management is quite large, our focus on automated risk identification in agile processes show some gaps.
There has been research into risk management of software projects. Some using automated risk models, some using machine-learning. The gap we detected was in managing risk on the smaller level of issues / stories. On the user story level research has been done in trying to aid in the effort estimation.
Risks are being identified on project level or on a more technical level. We see a gap where risks are detected on story or agile task level.
As shown in table 2.5 there is no clear algorithm that performs better than others. We couldn't find a shared approach on selecting the algorithm for the case of risk assessment.

Chapter 3

Risks

This chapter is dedicated to risk, through this chapter we hope to find a risk to use as a focus point to use in our experiments. To do this we have to find a risk that is relevant for the software engineering industry and which can be labeled on user story level. In doing this we hope to find an answer to our second research question. Our starting point is the use-case at the company which is to prevent risk in user stories. Because our approach is based in machine learning, risk labels are more important than indicators. We hope that the algorithm can discover the risk indicators if we can pass it labeled data. Risk is often measured on a project level, since stories are small chunks of work documenting risk on this level would cause a large amount of project overhead. This goes against one of the pillars from the agile manifesto "Individuals and interactions over processes and tools" [58]. Finding a source of risk labeled user stories is therefore unlikely.
Literature has identified a lot of risks over the decades in since this has been a research topic. Many of these risks that have been identified a long time ago are still relevant [29].
From this ranked set of risks, we focus on the risks with the highest rankings which can also be detected on scrum user story level. For the scope of this thesis focused on a single risk. The risk we chose to use for our classification purposes is "Focus on the wrong features", as it was highly rated in the survey. It also has built in labels in the form of story states we can use.
The rest of this chapter is structured as follows, we start with an explanation of the identified risks. This is followed by an explanation of our risk selection. The selection of risks was filtered first with help from experts from the host company. Which was then further filtered by executing a survey using the aforementioned risk definitions. Finally we explain on the risks, linking them to data from the agile process. In combination with the consequences they could have.

3.1 Initial risk identification

The scope of all project risks is large, for the scope of this thesis we only look at a specific risk. During our literature study we identified a number of papers listing risks that are relevant to software development, these have been listed in table: 2.4. Based on that research we initially identified 27 risks together with industry experts from the host company see table: A.4 in the appendix. We rated these risks following the method described by Li et al [28] leading to a risk rank of high, medium or low[1].
Based on ten highest rated risks we executed a survey to find out what the industry thinks is the most impactful from this subset.
The initial risk selection and rating was done by consulting with experts from the host company[2]. The complete set of risks was ordered on based on this rank. Within these levels of risks, ordering was done based on impact + probability. As this did not change the risk rank and led to more granularity, of ordering within high medium or low. From this we selected 10 with the highest rating.

3.2 Risk survey

We surveyed a group of 50 industry experts to rank these 10 risks. To give us an overview of how the industry views the relative importance between these risks. In the survey we requested respondents to rate the probability and impact of these risks using the scale defined by Li et al [28]. The survey was distributed through multiple channels, first within the host company's intranet. Second through directly sending it to software developers and sharing it on LinkedIn. Based on this the risk rank was calculated and the list was ordered. Which resulted in the following table 3.1.

[1]The ranking is based on an impact and probability, see tables [2.1,2.2]
[2]The team consists of experts from multiple disciplines with various levels of experience

ID	Risk	Impact	Probabilty	Risk
RSK005	Focus on the wrong features	4	4	High
RSK008	Scheduling too much work in a release	4	4	High
RSK004	Source-code containing hard to manage technical debt	3	4	Medium
RSK023	Missing hard deadlines from external software providers (E.G. APIs end of life)	4	3	Medium
RSK025	Modifying code with a lot of dependant code	3	4	Medium
RSK024	Introducing bugs	3	4	Medium
RSK015	Spending more time on tasks than estimated	3	4	Medium
RSK010	Missing knowledge to build a feature	3	3	Medium
RSK016	Losing members of the development team	4	2	Medium
RSK017	Planning dependant tasks in the same increment	3	3	Medium

Table 3.1: Survey result ordered risks

3.3 Risk details

Here we logically explain what the risk is, and how it might impact the project if it not handled. We also look at the what these risk implies on story level.

Focus on the wrong features

Having a focus on the wrong feature is a risk, because resources are being expanded on something that does not help the end user. This can happen due to many different reasons, some of which are harder to detect than others. Since we can't predict the future there is always a possibility of creating something which doesn't interest the end user. Another way of working on the wrong features is by working on features which are cancelled before they are completed. This is something we can detect on story level by looking at the state of a story. The end result of working on the wrong features is spending resources on something that does not add value to your product / project.

Scheduling too much work on a release

Scheduling is hard, when planning to little resources are unused and development time could be wasted. When scheduling too much some work won't be finished when it is promised. On story level it is hard to detect as story is part of the whole "release". However there are some indirect indicators: the amount of stories in a release and the amount of stories points associated with a release compared to an average release. This is an analysis of aggregated data, instead of reviewing the information on story level. The amount of work which makes sense for a release is dependent on the length of the release. The pressure of these release can both lead to a sense of urgency and increase productivity. But it can also add stress, disruptions and poor performance [59].

Source-code containing hard to manage technical debt

Technical debt, is a debt in a projects code base of code which often needs extra work before it can be modified. During feature development technical debt can cause slowdowns if the same areas of the codebase need to be changed. When commits are linked to stories, code analyses can be linked to these stories, based on the data in the commits. These analyses could be used to give stories technical debt scores. Models could then be trained to predict such a score for new stories. "The occurrence of TD is associated with a lack of progress and waste of time. This might have a negative influence on developers' morale" [60].

Missing hard deadlines from external software providers (E.G APIs end of life)

When a hard deadline is missed it could cause the software to become unusable for end-users. If basic authentication is phased out for OAuth2 in a system, but your system does not support OAuth2 yet before then. Users won't be able to sign into the service any more. The scrum process itself does not describe a way on how to document these types of hard deadlines. Tools and companies processes will therefore differ in how it can be detected. However NLP techniques could be used to detect deadlines in texts. Ideally the process would

contain some structured indication whether the deadline is truly fixed or not. These types of deadlines are usually known long enough in advance to plan for them. But when such a risk turns into a problem it can be very high impact to the user. Best case a part of an application or system doesn't work anymore, or in bad cases the whole system becomes unusable.

Modifying code with a lot of dependent code

Modifying code with a lot of dependent code is riskier than changing code which does not effect other parts of the system. The risk is causing unwanted behavior unrelated to the change someone wants to perform. By using change impact analysis techniques for a given method or class a usage graph could be generated. Once code is linked to a story, a score could be given based on the size of this usage graph. A modification doesn't have to cause problems, but if something is used in many places there is a higher likely hood of a bigger impact.

Introducing bugs

Bugs in software can cause problems for end users depending on the severity of the bug. However bugs can also be harmless without having any effect on the user experience. If stories are linked to commits, code analysis reports can show bugs. If DevOps pipelines are connected, build or test failures could also indicate bugs. The effect can be large or small, from crashing the software to bugs that won't impact operation at all. So the severity depends highly on the specific bug.

Spending more time on tasks than estimated

Estimating how much work certain tasks are can be hard. Sometimes tasks will take longer or shorter than estimated. If time worked is known for a story, it can be compared the estimated story points. If the time is larger or approaching the average story time for tasks with similar points, it is more likely to overshoot its estimation. If the tasks are consistently underestimated, it is likely that the project planning will not be met. It could lead to unhappy customers which are expecting to be able to use features. This in turn could lead to those customers looking for alternative products.

Missing knowledge to build a feature

If the knowledge of building something is not there, the developer(s) need to gain that knowledge first. In scrum this is often solved using "spike" stories, there is a limited amount of research into their effectiveness [61]. Many Scrum process management tools have ways of indicating a type for a story such as a research spike. But it is not specifically described in the scrum guide [62]. Spike stories are a common way of planning research new features before building them. If a story is marked as a "spike‘" the team can know that there is more uncertainty. Estimations on related tasks are harder and they could take more time.

Losing members of the development team

When losing people from the team, knowledge from that person is lost from the team. In the short team there are also less development resources for the team. Technical debt can have influence on morale [60], one way of looking at this is by checking stories for hard to manage technical debt. People leave for different reasons for leaving [63] Ma et al, researched data-driven measurement for measuring this risk. However their approach uses more information than what is available in singular stories. Due to the lost knowledge and resources, tasks might take longer and deadlines might be jeopardized.

Planning dependent tasks in the same product increment

Planning tasks that require the completion of other tasks in the same product increment. This like with many of the risks depends on the tooling as there is no fixed process for handling this. Jira allows for linked issues with an indication of blocks and is blocked by[64]. The tool used at the host company uses a field is_blocked, and fills in they number of blocking stories in a field called blocked_reason. It lowers flexibility of the development team by requiring certain tasks to finish first. If the first tasks take longer so do the dependent tasks which can cause planning problems.

3.4 Risk causality

Risks do not exists in a vacuum and can influence each other. We spent some time with industry experts to identify relationships between risks. Specifically for the risks described in the previous section, this is shown in figure 3.1.

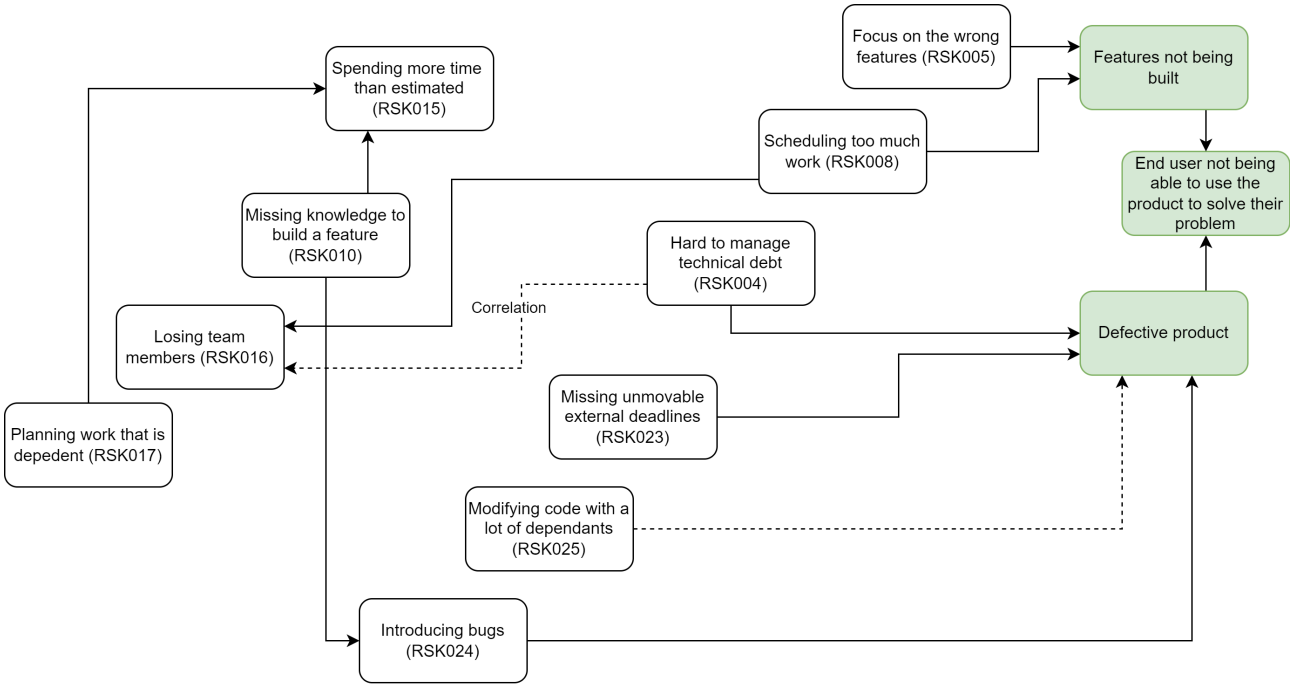

Figure 3.1: Risk causality

3.5 Risk focus

The risk we choose to user during our further experiments is “Focus on the wrong features”. Because it was rated the highest together with the risk of scheduling to much work in a release. However, our research looks at the story level and omits the scope of a whole release. So to answer the second research question “what kind of risk labels can be found in user stories”. We found that user stories “states” can be used to infer a focus on a wrong feature. This could lead to users switching to different products as a competitor could be building what they need.

Chapter 4

System Design

In this chapter we will explain our approach to creating this system. The starting point will be the use-case of the host company. We start by explaining the goal and challenges of building such a system. Followed by the requirements of the system. Based on these requirements we will explain the architecture of the system. Following the design we will explain the implementation. Starting with technical considerations we had to make, finally we will describe the details of the systems components.

The goal of our system is to improve agile processes (Scrum) using automated risk prediction. We hope to aid the continuous feedback loop of risk management which was found to be beneficial in agile settings [19]. The system is designed without any specific risk in mind, however for our implementation we choose a specific risk, due to time constraints. As the models need to be trained for a specific risk, which requires a data set which is specific to that risk. As explained in the previous chapter the specific risk will be "focus on the wrong features". Because the it's ranked importance by industry experts and available data labels.

4.1 Challenges

We will start with the technical challenges that lie ahead. The key problems that our design needs to solve for a system that will help us predict risk.

4.1.1 Process differences

There are many processes that can be followed to create software, but even when looking at a single framework such as Scrum. There are many things not described in the framework, leaving it up to interpretation. This leads to engineering logs that don't follow a fixed format and differ wildly between companies and sometimes even teams. Often tools that can be used for monitoring the software development process such as Jira or Azure DeVops, can also be customized by the users to fit their process. Adding another possible step of variance in the data. Therefore we cannot create a one-size fits all solution without losing some resolution in the data. As we would have to simplify the data from more complex processes into something that is compatible with simpler processes. This data is not often labeled for usage in risk analysis, which might make it hard to use for classification purposes.

4.1.2 Training data retrieval

Data from agile processes is commonly saved into Issue Tracking System these systems are aimed at project management activities, not necessarily for extracting data for the use of ML. Most of these systems allow for data exports, but it is unlikely that the output of different systems will be the same.

4.1.3 Text processing

Machine learning algorithms often require a numerical values instead of raw text, a processing step is required for those algorithms before they can interact with text. This is not the case for all algorithms.
User stories can be written by different people, when extracting risk we don't want to the writer's style to impact the risk prediction.

4.2 Requirements

In this section we will explain the requirements of the system. The requirements where gathered based on the use-case of the host company.
The system has multiple parts with different users in mind. It can be split up into three main parts.

1. Data retrieval
2. Training
3. Predicting

The end goal is to be able to give predictions on user stories. For that purpose we need ML models that have been trained. To train the models we need data.
We started with our requirements gathering by looking at the existing process and consulting with experts at the host company. Combining this with the challenges, we identified a set of functional and non-functional requirements.
Our system has three main stakeholders, they have different roles relating to the system.

- Scrum Team
  - Product Owner (PO)
  - Development Team
- Classification developer

Due to the variety in tool usage across the industry a key requirement is flexibility's. The system should not expect the users to be using a specific Issue Tracking System. Therefore exported data from various ITS should be usable without changes to the system.
Because the risk domain is very broad, software development projects have different requirements in risk management. Fixing the risk which the system can predict limits the projects to which it can be applicable. The risk which is being predicted should therefore be configurable.
An important part of the engineering logs captured in the agile process is natural language. Before this can be used by machine-learning algorithms, it needs to be processed into a suitable format often numerical [65, p. 4]. But the exact format depends on the chosen algorithm.
Since literature has not found an algorithm which has "solved" risks prediction tasks. The system needs to be able to compare machine learning algorithms to find an optimal modal for the user. The system should help the user with making a decision in selecting a algorithm. Depending on the amount of hyper parameters that where compared between algorithms this can lead to a large amount of data. An overview of this data should be presented to prevent overwhelming the user. While allowing access to the data for deeper analysis if required, for better understanding of the chosen trained model.
As machine-learning is a rapidly evolving field, the system should be modifiable [66, p. 282]. To allow new ML algorithms to be added later if they are found to be highly suited to risk prediction tasks.

4.3 Architecture

In this section we will explain the architecture of our system, which components there are and how they interact. Another goal of the architecture is to create a bridge between requirements and implementation. We start with an overview below in figure 4.1, after which we explain these components and their subsystems.

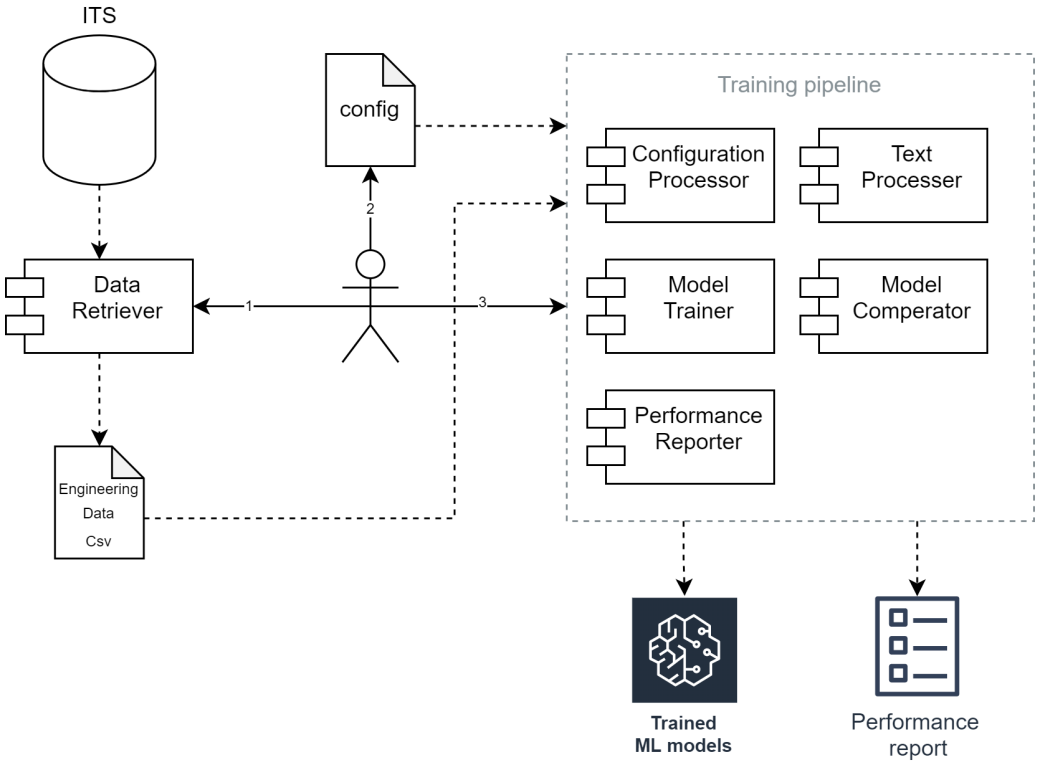

Figure 4.1: Component diagram

4.3.1 Data retriever

The first component will be specific to the user of the system. Because the data retriever interfaces with a development team's Issue Tracking System and its unique setup. It's task is to take the data from the agile process and transform it into a CSV file. For this data set a configuration file can be created so the training pipeline will know how to process the input.

4.3.2 Training pipeline

Since the literature does not agree on a single best algorithm to use, we need something to find an ideal algorithm for the risk we want to classify. Because combinations of certain data sets and features will have different results when used as input for the ML algorithm. The algorithm also have different hyper parameters[1] which can be tuned to change the performance of the model. The training pipeline will therefore have to compare algorithms. In figure 4.2 we show the flow of data from the user to the system and back for the training pipeline.

[1]Hyper parameters are used to influence the learning process of the algorithm and are separate from the data derived during training

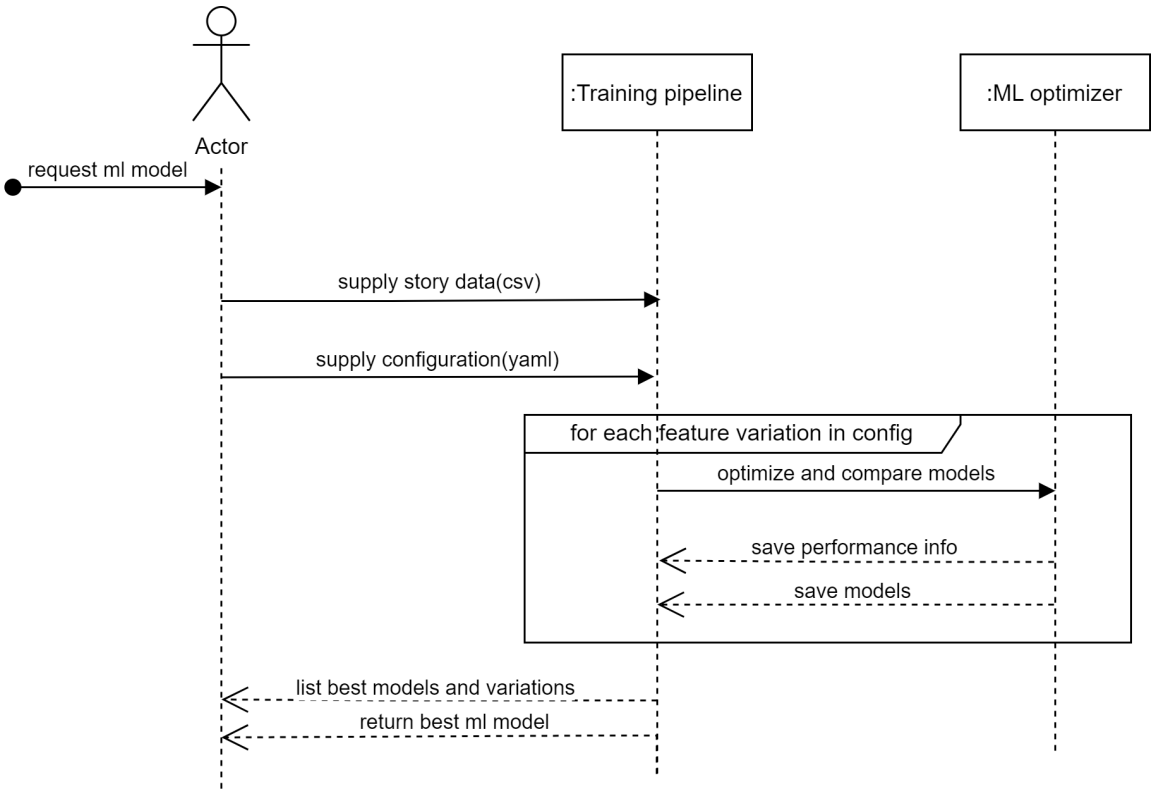

Figure 4.2: Training pipeline system sequence diagram

Configuration (processor)

Our system has the intent to help practitioners to create predictors for risks. We don't want to hard-code what input data is used for classifications. As this adds dependencies to specific issue tracking systems, or requires the user to create mapping schemes for their data onto what is required by the system. Therefore the most important things that can be configured in our system is the following:

- Input columns
- Prediction column
- Classification labels

First what the names of the columns are, that will be used as input features for the machine-learning algorithms. Second which column the machine-learning models should be predicting, if a labeled data set is available this would be your label column. Additionally it would be useful be able to have configurations options for the text processing component. Different teams have their own ways of communicating, by giving the user flexibility on the text pre-processing we hope to achieve better results. This being configurable will also allow us to explore the first research question.
There needs to be a part of the system which loads the configuration and helps the other subsystems to behave according the user's configuration.

Text processor

User story's are described in natural language, unstructured data not in structured columns. Without the usage of NLP a lot of the information in the story would be ignored when doing predictions.

Performance reporter

The goal of this is to give the user of the system insight on how the various ML models performed against each other on configured data sets. The comparison data we will generate is tabular, rows for each tested configuration. Columns for its performance and parameters that where used. For more thorough analysis we want to save this tabular data in CSV format. So it can be easily analysed in many existing tools. We also want graphical representations to be able to do visual comparisons. We need some overarching document which can contain multimedia for the graphs and tables.

4.4 Technical considerations

In the coming sections we will focus on the implementation of the architecture. First we have to decide on a technology stack to build our system on. There are many possible ways of implementing NLP, we have to decide what we will implement. Since we are going to compare ML algorithms, we will not implemented them from scratch as it would take too much time and is more error prone. We have to decide on a framework or library to use for these implementations.

4.4.1 Language selection

At the core of a technology stack lies the programming language(s) that will be used. This choice has impact on what available libraries and frameworks we can use. Since the core part of our system is related to machine learning we started with a search on github to get a feeling of the commonly used technologies.

Language	Repository count
Jupyter Notebook	164,562
Python	80,960
HTML	25,547
MATLAB	18,568
R	8,668
Java	5,139
JavaScript	5,076
C++	3,026
C#	1,951
TeX	1,634

Table 4.1: Github Search: "machine learning" [67]

The most common technology used shown in table 4.1 is "Jupyter Notebook". This is an interactive, highly visual way of interacting with python code, which is the second most common repository language. With a larger amount of users, better support is likely. Since we want our system to be able to run headless we will choose python of Jupyter notebooks. While this takes away an easy way of displaying results to the user, it allows the system to run autonomously without requiring a user present.

4.4.2 Natural Language Processing

Due to the textual nature of user stories, our system requires text processing. Our first research question asks what kind of pre-processing is required, this is dependent on the ML algorithm which is used. One of the challenges identified in the system design is about human writing style. Different people can write stories, while their meaning could be similar the writing style could make it hard for the model to find similarities. To prevent this, before we can process the text it needs to be normalized. Many ML algorithms require input in a numerical format, however there are algorithms that are made to work with text directly. To support ML algorithms that do require numerical inputs we propose the the usage of NLP techniques. Different ways of processing text have different advantages and disadvantages. We will therefore implement multiple processing methods. Our system will do this in two main ways, bag of words (BoW) and feature engineering.
The bag of words technique looks at the texts in the training set and creates a feature for each word, where the value is the amount of occurrence in a specific record. However, a large amount of features can lead to models overfitting on the data and will not be able to process words that are not part of the training set.
The other way we interpret text is by using other means of analyzing the text, the results of these analyses are then passed to the algorithms. We use a model for sentiment analysis which creates a single column. We allow for the configuration of zero-shot analysis, where the user can add their own analyses as they see fit. Zero-shot classification allows us to pass both a set of labels and text to the model. The model then tries to understand both the labels and the text, then it will match the text to best fitting label. For this we use the pre-trained language model BART [68]. Because it is pre-trained and takes both text and labels as input we can allow the user of the training pipeline to choose labels that can more directly target their texts. An example of this is configuring a generated column "urgency", where the labels would be "urgent", "moderate", "not urgent".
The last way we do feature pre-engineering is Latent Dirichlet Allocation (LDA)[2]. This is an automated way

[2]Not Linear Discriminant Analysis which is a form of dimension reduction

of doing topic modelling, it extracts n topics from the corpus of text it is passed[69]. It can then predict the probability of those extracted topics in rows of your data. We then use the probability of these topics as input for our algorithms.

4.4.3 ML Framework

There are many good choices for ML frameworks widely used ones include:

- Tensorflow `https://pypistats.org/packages/tensorflow`
- (Py)torch `https://pypistats.org/packages/torch`
- Keras `https://pypistats.org/packages/keras`
- Scikit-learn `https://pypistats.org/packages/scikit-learn`

name	Downloads day	Downloads week	Downloads month	retrieved
scikit-learn	885,385	5,578,789	23,291,468	2021-10-15
tensorflow	456,324	2,796,192	12,082,474	2021-10-15
keras	229,882	1,426,366	6,018,914	2021-10-15
torch	220,984	1,321,159	5,207,583	2021-10-15

Table 4.2: Python ML lib downloads

In our implementation we made use of the scikit-learn framework[3]. It supports a large amount of algorithms for regression, classification and clustering. The framework also has dimension reduction features helpers for data pre-processing and metrics about the accuracy and confusion of trained models. It is open source and can be used commercially through its BSD license. Another advantage is a common API for different machine-learning algorithms, making the framework easier to learn and allowing for easy reuse of code for different algorithms.

4.4.4 ML algorithm selection

Different types of input data can have can have large effects on the performance of the final model. With the large amount of variance in software processes leading to different engineering logs, this can lead to wildly different results. We therefore implemented multiple algorithms so they can be compared.

K-Nearest Neighbour

This algorithm classifies data based on proximity of K neighbours, it is a relatively straightforward algorithm. However, it can have poor accuracy on data with outliers and the computational complexity grows with the size of the data set. It requires the user to select a value of K.

Naïve Bayes

Is a statistical algorithm based on the Bayes probability theorem, the naivety refers to the fact the algorithm assumes independence of input features. Which makes it a better fit for categorical data instead of numerical. It can perform strongly if the data is independent.

Multi-layer Perceptron

This is a neural network, unlike the previous statistical algorithms therefore it is a black-box which makes it harder to interpret. It is good with modeling non-linear data and can execute predictions fast at the cost of a potentially heavy training cost.

Decision Tree

Decision tree's are relatively simple which make them easier to interpret and explain. DT's require less data pre-processing, however small changes in the training data can cause large changes in the tree. Which can cause instability based on light changes in data, this algorithm often takes more time to train compared to other algorithms.

[3] `https://scikit-learn.org`

Stochastic gradient descent

This is a very efficient algorithm and handles large amounts of data well. It is sensitive to outliers and works better when data is scaled.

Support vector machine

These are not very suited to large data sets or overlapping classes. With a lot of data it can take a large amount of time to train, the algorithm requires feature scaling to perform well. It is stable as small changes in the data will not have big effects on the output. The SVM can also handle non-linear data well.

4.5 Components

In this section we will dive deeper into the components described in the previous chapter. The order of the components is described in the order in which the user / system would interact with them. Starting with data-retrieval, followed by configuration for the pipeline finally how the training pipeline works and how it's results are presented to the user.

4.5.1 Data-retrieval

We implemented the data-retrieval component twice, once for the host company to retrieve proprietary data. A second time to retrieve data available from public Jira repositories.

Host company data

The host company data comes in two parts, git logs and story data which are linked through a unique identifier that prefixes every commit. However these logs are stored in two different places, so we need to retrieve them and join them so they can be easily fed to the machine-learning algorithm.
The second part is the story parser, the user stories are saved on a cloud platform. The stories there are linked to other related pieces of information.
The engineering logs we have available from the host company are from the following sources:

- Scrum logs
- Git commits

Scrum process info The host company uses the Scrum framework for their software project methodology and use an internal tool to log the Scrum process. This tool stores the following:

- User Stories
- Theme
- Epic
- Feature
- Sprint
- Release
- User

The system that is used exposes most data through an REST API. We created a simple script takes a time-range as input, and retrieves all user stories for that time range and outputs it in CSV format.

The version control system that is used at the host company is git. The company follows a process where git commits are written in the following form "ASTRY00001: updated some feature". So commits can be traced back to their requirements, this does not hold for merge commits as branch squashing is not used. We need to process these git logs into a format that is easier to interact with. So we are not dependent on having the actual repositories where we want to use this data. To do this we can use git's built in "git log –format". Combining this with a shell script using AWK we can prepend this with the folder name, so we know from which repository the commits came from. This script can be found in the appendix under listing B.3. This allows us to create a single CSV with the commits for all projects, which should help with finding commits by story number (unique identifier).

Jira data

We want to create an open data set so comparisons of similar techniques can be done using the same data, which we hope allows for clearer comparisons between techniques. In our research we used two data sets, one

proprietary containing user stories from the host company. The second dataset is public we will explain how it was retrieved and how our method could be used to retrieve similar data sets.
Choetkiertikul et al [20] published a data set, with their research on story point estimation. Unfortunately only it contains the "issue key, title, description and storypoints" fields. So for our usage this data lacks risk labels, however this is a good filter on all the public projects available on Jira. Using a script we tried to identify which project contained the most data from their list[4] of selected projects. From this we identified lsst-corp's "Data Management" project[5] which contained 4667 rows in Choetkiertikul's repository. At the time of writing the project on Jira contains 31724 issues. Jira has a CSV export feature, however this is limited to 1000 rows of data. Due to this limitation we chose to use the Jira REST API to retrieve stories using pagination to get around the 1000 records limit. We created a Node.js script[6] which can easily be modified to retrieve data from other public Jira projects. Due to Jira's customizability it is likely that a mapping function we created needs to be modified to retrieve fields specific to that project. In appendix C we display the structure of the API response and it's types for a given Jira story, this to illustrate the necessity of an extra data mapping layer. After retrieving these 31724 issues, we applied the following filters: Status can only be Done, Invalid or Won't Fix. The type of the story should be one of Bug, Story or Technical task, resulting in final set of 23715. These specific statuses where chosen because only the final states are important, detecting that something will be in progress doesn't tell us whether it will end up being resolved.

4.5.2 Input (Text) processor

Machine-learning algorithms generally use numerical input values, however this depends on the specific algorithm. Furthermore "Many machine learning algorithms perform better when numerical input variables are scaled to a standard range." [70]. To turn user stories into scaled numerical values we need to execute a few processing steps. In the configuration we can either choose a bag of words or feature pre-engineering. However before we pass the text to those components we need to normalize the text, so texts with the same meaning with minor variations can still be matched together. The first step is turning the text into lower case, this is followed by stopword removal. Stopword removal is a common pre-processing step which commonly occur across all texts in the corpus. These words help human readers understand the text, but make the input texts similar for the algorithms. This is followed by a technique called stemming, to do this we use the NLTK library [71]. By stemming we reduce the words into simpler forms to get more matches on words with similar meaning. For example the words "planning, planned" would both get stemmed to the word plan, now we can pass this normalized text to the processors. The bag of words technique looks at the texts in the training set and creates a feature for each word, where the value is the amount of occurrence in a specific record. However this can lead to a large amount of features which can the models to overfit on the data.
The other way we interpret text is by using other means of analyzing the text, the results of these analyses are then passed to the algorithms. We use a model for sentiment analysis which creates a single column. The system also incorporates configurable zero-shot analysis. Zero-shot classification allows us to pass both a set of labels and text to the model. The model then tries to understand both the labels and the text, then it will match the text to best fitting label. For this we use the pre-trained language model BART [68]. Because it is pre-trained and takes both text and labels as input we can allow the user of the training pipeline to choose labels that can more directly target their texts.
The last way we do feature pre-engineering is Latent Dirichlet Allocation (LDA)[7]. This is an automated way of doing topic modelling, it extracts n topics from the corpus of text it is passed[69]. It can then predict the probability of those extracted topics in rows of your data. We then use the probability of these topics as input for our models.
As values in a standard range usually perform better [70], the last step in the processing is scaling the numerical ranges into standardized ranges. This is done using scikit-learn's ordinal encoder.
With this component we have partially answered our first research question, "what kind of pre-processing is required on engineering logs for usage in risk predicting machine-learning models?". Due to the textual nature of project information, mainly text processing. We will still need to evaluate which method performed best for data sets.

4.5.3 Configuration Processor

Looking back at the challenges, one of which was software process variance. Which leads to process data variance, we don't want our system hard-coupled to a specific ITS. Therefore our system should be able to handle various data sets. The system needs to know how to handle this data so we have to apply reasonable

[4] https://github.com/jai2shukla/JIRA-Estimation-Prediction/tree/master/storypoint/IEEE%20TSE2018/dataset
[5] https://jira.lsstcorp.org/projects/DM/issues/DM-32583?filter=allissues
[6] https://github.com/DriesMeerman/Risk-Assessment/tree/main/code/javascript/jira-retrieval
[7] Not to be confused with Linear Discriminant Analysis which is a form of dimension reduction

limits. We therefore choose CSV files as labeled data sets, since these are commonly used in machine-learning. The configuration of the training pipeline is defined in a YAML file. We chose YAML as a configuration structure due to it's simplicity in writing and not requiring a lot of boiler plate. Other options where JSON or INI files, the former requires correctly placed quotes and commas. Which is not the most user friendly when creating the configuration. INI files on the other hand, are even simpler to write, but lack features like nesting and lists.
The most important thing we need to configure are labels and inputs. The ML algorithms need to know what data will be used as input and which column are the labels. A key called "y_column" is used to indicate the name of the labels. Multiple variations can be made of the text fields that will be processed, allowing the user to compare different data inputs to train the models. Since some fields might be reused throughout multple variations they can be re-used and only have to be defined once in the configuration. An example of this is shown in listing 4.1.

```
y_column: "status"
variables:
  base_fields: [points, sprint_count]

feature_variations:
  description:
    x_cols_vars: [base_fields]
    vectorize_fields: [description]

  summary_and_desc:
    x_cols_vars: [ base_fields ]
    vectorize_fields: [ summary, description ]
```

Listing 4.1: Data configuration

Multiple ways of processing text can be used by the system, this is another thing that needs to be configured. Firstly which fields in the data set, need to be processed as text. Because numerical data such as user story points (effort estimation) is not text and should not be treated as such. If the user wants to use bag of words then the configuration is mostly finished. However if the user wants to use feature pre-engineering, the system requires labels for zero-shot analyses. An example of how such a zero-shot analysis config can be found in listing 4.2.

```
zeroshot_analysis:
    -
      name: urgency
      labels: ["urgent", "moderate", "not_urgent"]
    -
      name: complexity
      labels: ["simple", "complex"]
```

Listing 4.2: Zeroshot configuration example

Full example of the configuration can be found with the code on github[8].

4.5.4 Model training and comparing

In the implementation the training and comparing are a bit overlapping because of performance reasons. We don't want to keep trained versions of each models in memory while training new models so we can compare all trained models at once. Instead the train test data split is done at the start, all models are trained and tested on the same set to keep the comparison equal.
Our implementation uses scikit-learn which has similar API's for its models which makes it easy to compare them. However. modifyability was one of the requirements to allow algorithms to be implemented in the future. We can't know whether such algorithms especially for risk related tasks, will be implemented in scikit-learn. Therefore we created our own wrapper class that the pipeline can interact with abstracting away the details of the models. This gives us the flexibility to allow algorithms to be implemented in other libraries while still being compatible with the model comparator. We call this base extension "ModelTest" and it can be extended for each machine-learning algorithm.
This is shown in figure 4.3.

[8] https://github.com/DriesMeerman/Risk-Assessment/tree/main/code/python/training_pipeline/configs

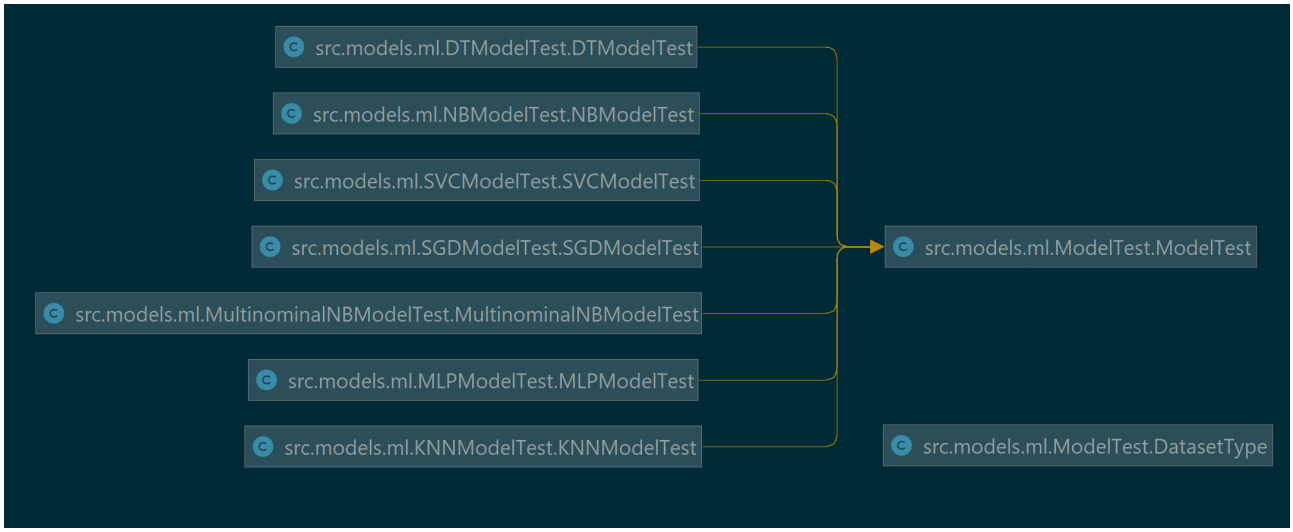

Figure 4.3: Extended Training pipeline ML wrapper classes

4.5.5 Performance reporter

The performance reporter component has multiple responsibilities. Firstly it needs to save the performance of the machine-learning models, so the models can be compared and the most suitable candidate can be selected. This data needs to be saved, so the user of the system has the option of doing a deeper analysis of all the models that where compared. To increase the usability we opted to generate a summary, containing the models with the highest accuracy in a table and graphs showing the differences. This is shown in figure 4.4.

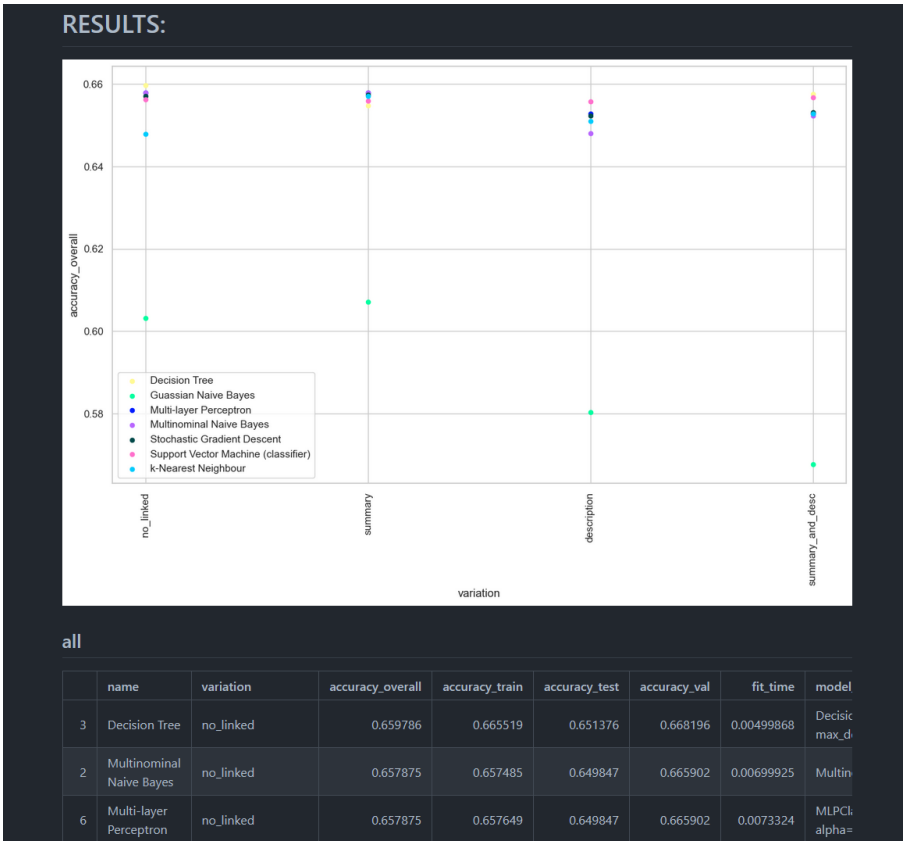

Figure 4.4: Snippet from the generated performance report

Lastly since this component has access to all the models and is already heavily involved in IO[9] operations. We added the persisting of the models here, with file names that correspond to their configuration. We also added the models accuracy in the filename, so the best performing models could be easily detected.

[9]Input / Output, writing data to persistent storage in our case

Chapter 5

System evaluation

We now have a system to compare different machine learning algorithms for risk prediction. Which we can use to find a suitable trained model, to implement risk predictions on user story data. The code base, configuration and data which was used can be found here: `https://github.com/DriesMeerman/Risk-Assessment`.
The experiments will be using the data to classify this risk in the following form. A story which does not get completed was work with a wrong focus. The classifications in our experiment will reflect this by trying to predict one of the following end states:

- Cancelled
- Completed
- Stale (Neither of the above, not updated in the last 8 weeks)

We will first explain the experiment design followed by the experiment results.

5.1 Experiment Design

The goal of the experiment is two fold, we want to know how different algorithms and different sets of data impact the results. To be able to answer RQ3 we need to train a model we can use, and we need to know how well it will perform. Specifically subquestion 3.1 "how do different ML algorithms behave when using user story data as input?" For this we have the following hypothesis: *Different ML algorithms will have different levels of accuracy* and *The selected set of features will have an impact on the accuracy of the ML algorithms.* For this experiment we first have to create configurations for the data sets we want to use. Then we can execute the pipeline which will train the machine-learning models and generate performance statistics and finally we will compare the results. The variables that will be tested are two different data sets, different splits of input features in those data sets and the previously described machine-learning algorithm.
The **configurations** contain two components the feature variations and the text pre-processing information. These different configurations will allow us to compare models trained on different subsets of the data-sets. With the differently configured text processing we hope to find a way of text processing which is suitable for user story information. Another difference in feature variance is based on the fact that the data sets come from two different companies using different tools and processes. Using the same configuration would lead to the pipeline trying to access data which is not available in the set. However the same heuristical approach was taken, the textual fields contain the most important information we create different feature variations based on them. In both data sets a summary and description type field was available, other feature variations were created based on the expected availability of columns. For example in the proprietary data set, we had GIT commit related information. This might not always be available so we where interested in seeing how the models performed with and without this information. In total there will be 4 configurations, two per data set distinct ones for the different types of language interpretation. In the configuration for the engineered features we enabled sentiment analysis and used zero-shot analysis for the following topics: "urgency, general_topic, se_topic, type_topic, clarity, buzzwords, complexity, emotion, test-ability, dependency and other_sentiment" [1].
Once the configurations where created we could upload the data sets to a server and execute the pipeline using the following command[2] (listing 5.1).

```
python3 src/pipeline.py config_jira_bow.yml > output/raw.txt 2>&1
```

Listing 5.1: Pipeline execution

The command starts the process in the background and writes the standard and error output into a file. The relevant output the pipeline itself creates is saved separately these logs are only used for debugging purposes in case the system fails. The pipeline will then create performance statistics both in a tabular format and graphs showing comparisons between the variations and algorithms.

[1] See `https://github.com/DriesMeerman/Risk-Assessment/tree/main/code/python/training_pipeline/configs`
[2] Command executed in a Tmux environment to prevent shutdown of the process without a connected user

5.2 Results

When we used the Jira data set with the BoW technique of processing text, the best performing model was K-nn with an accuracy of 66.8% see figure 5.1. With the ML pre-processed data the best performing model was the decision tree had an accuracy of 65.9% as shown in figure 5.2. For this set the performance of many models was quite similar, with the guassian naive bayes having the lowest performance. Which makes sense since the input features are related by virtue of being part of the same user story.

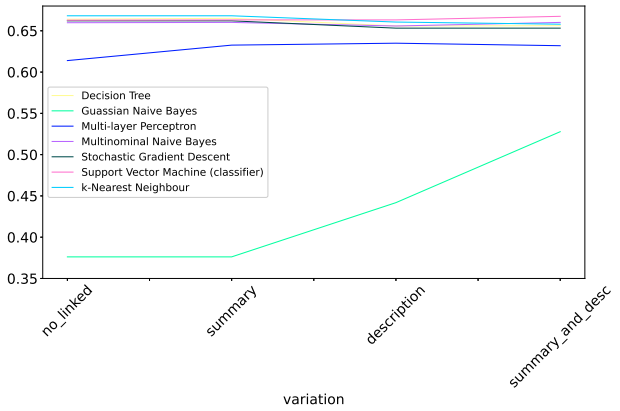

Figure 5.1: Bag of words (Jira)

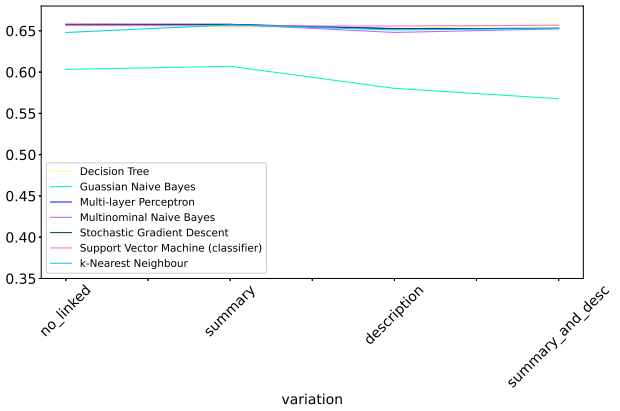

Figure 5.2: ML pre-processed (Jira)

The results for the 4Industry data set using BoW in figure 5.3 seem better at first, with many models reaching a near 100% accuracy. However the models seem to have overfitted on the data. The best performer with a more realistic accuracy is Multinominal Naïve Bayes with 67.5%. Any model with a performance over 70% we assumed to be overfitted when looking at the other results. The best performance from one of the sets that did not use any natural language processing[3] was a decision tree with 61.8% accuracy. Gaussian Naïve Bayes has the lowest accuracy again. In the ML pre-possessed output we see a distribution closer to the public data. As shown in figure 5.4, due to the more limited amount of columns it is less likely to be overfitted. In this case the best performing model was the Multi-layer Perceptron, with an accuracy of 61.6%.

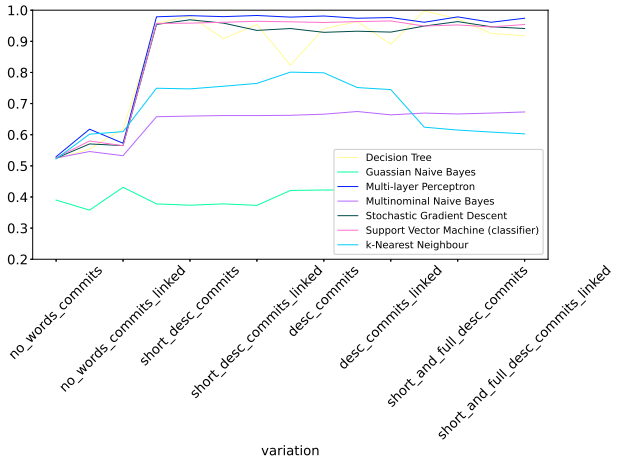

Figure 5.3: Bag of words (4Industry)

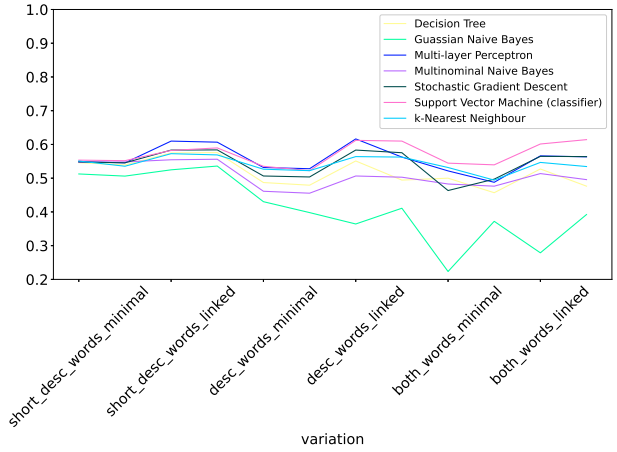

Figure 5.4: ML pre-processed (4Industry)

The fact that these models score so close in terms of accuracy can mean that there is a large set of stories which are easy to predict. While the other stories are hard to predict, causing the models to correctly predict the same set of data.

Looking at these results we can now answer **RQ1**. The data set which is used will influence what the most effective way of pre-processing the data is. In the Jira set there was not a very large difference in performance, two models (GNB and MLP) did better when using pre-processing the other models scored similarly. Looking at the 4Industry data set a big difference can be seen. Using the bag of words technique, caused some models to get unrealistically good results. The k Nearest Neighbour model performed within range of what is expected, but since multiple other models have overfitted it seems likely that this could have happened for this model as well.

[3]Which where unlikely to be constrained by the BoW overfitting problem

Using bag of words could in some cases lead to better results, but seems less stable and harder to validate. In cases where we find the results more trustworthy the feature pre-engineering seems to get better results.

Chapter 6

Ensemble approach

In this chapter we will try to improve the performance of our generated prediction models using ensemble algorithms. Machine-learning ensembles are algorithms that combine several base models to achieve better results. Ensembles can increase the performance (the accuracy of the predictions), they can also increase the robustness (reliability of the average performance). Similarly to the single machine-learning models we don't know which one will perform well. The goal is to improve over the singular algorithms, we will therefore compare multiple ensembles algorithms to which has the best performance increase.
The rest of this chapter is structured as follows, we will start with details surrounding the ensembles followed by their implementation. Then we will execute two experiments one to compare the ensembles to the single algorithms, followed by a comparison with human risk prediction.

6.1 Ensembles

Our system compares the following ensemble methods:

Adaboost, this uses sequentially trained weak learners to get better predictions than the base estimators.
Bagging, it splits the training data into multiple sets and trains models on subsets of this data. Predictions are done using a weighted average of these models.
Random forest, this is often seen as a extension on bagging, as it uses Decision Trees as weak learners on random data subsets.
Gradient boost, is another boosting technique where Decision Trees are used to correct errors of the previous stage of training. They can take longer to train as it happens sequentially, they can lead to better accuracy than random forests but are also more likely to overfit.
Stacking algorithm this is a meta algorithm which uses predictions of base models as inputs. From the inputs of its base models it will intelligently decide which models input should have priority when giving the final prediction.

The ensembles algorithms, like the singular algorithms where implemented using scikit-learn. Executing training the ensembles happens in the same way as training the other models. By setting the "ensemble" flag to true in the configuration, the ensembles will be used. Our hope is to achieve higher accuracy with ensembles than with the individual models.

6.2 Implementation

The existing "ModelTest" class has been extended for usage with ensemble algorithms. We have also extended it for the use of ensemble algorithms, since they have some other shared properties an abstraction for that is made called "EnsembleModelTest". The new structure is shown in figure 6.1 Since the class was extended from "ModelTest", the interface for training and comparing remains the same. Requiring us to only implement the the specific ensembles without requiring changes to the core of the system.

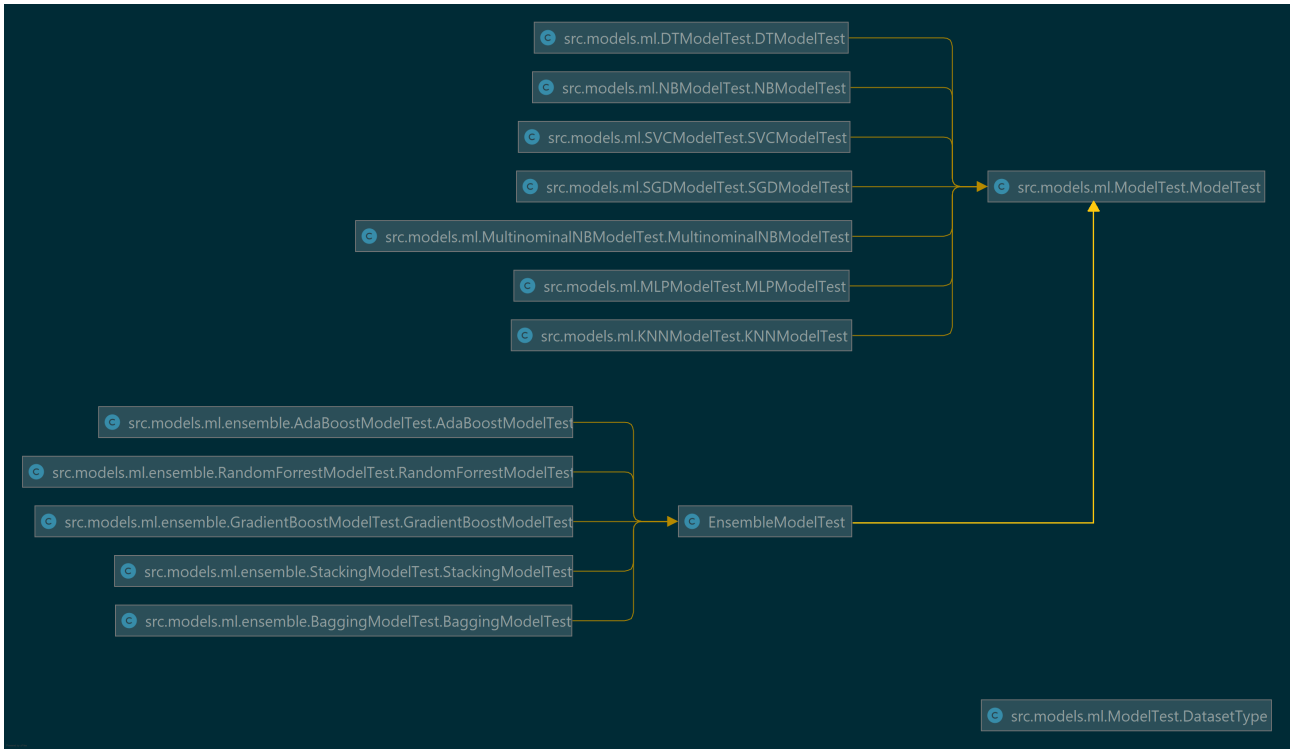

Figure 6.1: Extended Training pipeline ML wrapper classes

6.3 Ensembles algorithms compared to single algorithms

Because the ensembles are treated the same as the regular algorithms, we can now rerun the first experiment, on the same data using the same text processing techniques. By doing this we can do a direct comparison of the ensembles against the singular algorithms.

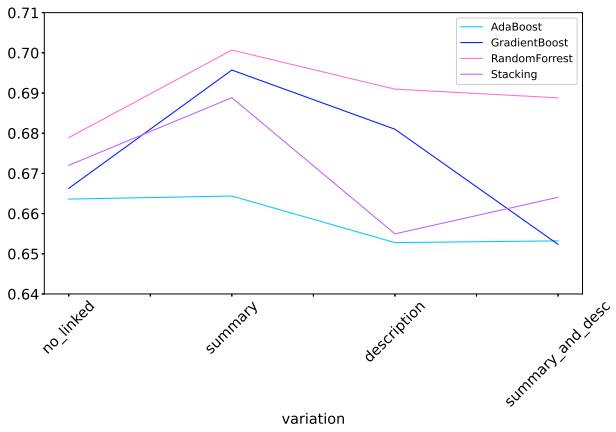

Figure 6.2: Ensemble BoW (Jira)

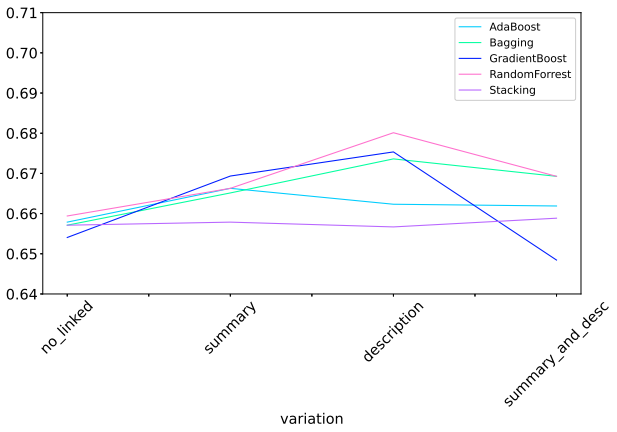

Figure 6.3: Ensemble pre-processed (Jira)

The best performing ensemble model for the Jira data set was the RandomForrest ensemble, achieving 70.1% accuracy using BoW and getting 68% with ml pre-processing. Bag of words, scored higher overall but also had a higher variance in the Jira set compared to the engineered features see figures 6.2, 6.3. There where 9 ensembles out of 20 that outperformed the best performing model (K-NN). When looking at the pre-processed set, the lowest scoring ensemble, gradient boost was still close to the best performing non-ensemble model with 64.8% accuracy. Out of the 20 tested variations for the pre-processed set, 12 ensembles performed equal to or better than the Decision Tree.
For the 4Industry data set when using BoW, we see overfitting occurring again. Comparing the results for the variations that did not use text fields we see that, there where 5 variations with scored a higher accuracy. The highest being RandomForrest which achieved 66.5% accuracy. With the pre-processed text the best achieved result was an accuracy of 68.0% with the Random Forrest ensemble. Compared to the MLP with 61.6% this is a good improvement.

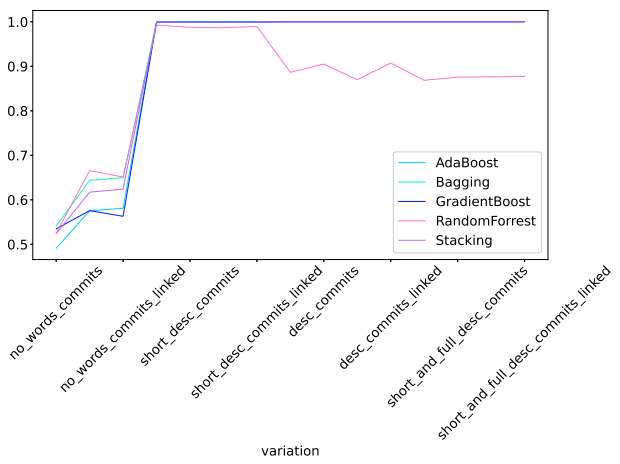

Figure 6.4: Ensemble BoW (4Industry)

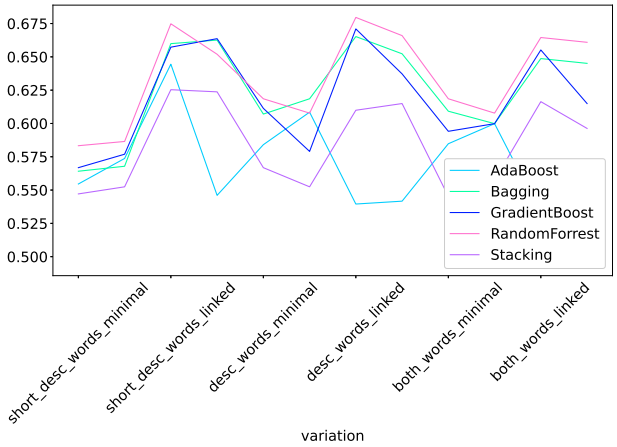

Figure 6.5: Ensemble pre-processed (4Industry)

6.4 Ensembles compared to Human risk detection

This section compares the performance of our best models to a human baseline. We start by explaining how this baseline was recorded, followed by its results which are then compared to the results from best performing models generated by our approach. In table 6.1 we show metrics related to the f1-score for the classifications executed by people. The first column shows precision is the percentage of correctly predicted values in that class, "Done" was predicted correctly 25 out of 37 times (67.6%). The other 12 predictions where "false positives" identifying a different class as "Done". Recall shows the correctly predicted classes compared to "false negatives", for "Done" this is 25 out of 60 (41.7%). The f1-score shows the harmonic mean between the precision and recall.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

[1] Finally the support column shows the amount of answers in the dataset with that classification. Accuracy shows the average amount of correct answers in total, the macro average shows the average of the f1-score per class. The weighted average does the same as before but takes the support into account.

	precision	recall	f1-score	support
Done	0.676	0.417	0.515	60
Invalid	0.367	0.550	0.440	20
Won't fix	0.087	0.200	0.121	10
accuracy			0.422	90
macro avg	0.376	0.389	0.359	90
weighted avg	0.542	0.422	0.455	90

Table 6.1: Human f-score

[1]True positive (TP), false positive (FP), false negative (FN).
[2]Best performing model Decision Tree, for summary and description feature variation
[3]Best performing ensemble random forrest, for description feature variation

	precision	recall	f1-score	support
Done	0.718654	0.947581	0.817391	744
Invalid	0.311111	0.084848	0.133333	165
Won't Fix	0.388889	0.201646	0.265583	243
accuracy			0.666667	1152
macro avg	0.472885	0.411358	0.405436	1152
weighted avg	0.590722	0.666667	0.603017	1152

Table 6.2: f-score Jira pre-processed [2]

	precision	recall	f1-score	support
Done	0.710808	0.961792	0.817469	759
Invalid	0.545455	0.151899	0.237624	158
Won't Fix	0.493827	0.170213	0.253165	235
accuracy			0.689236	1152
macro avg	0.583363	0.427968	0.436086	1152
weighted avg	0.643867	0.689236	0.622828	1152

Table 6.3: f-score Jira pre-processed ensemble [3]

Chapter 7

Discussion

In this chapter, we discuss the results of our research and experiments on machine-learning algorithms and ensemble algorithms. We start by looking at the required data pre-processing, first the transformation required on input data. Second, the way we retrieved a labelled data set for the task of risk prediction. Then we look at the performance of single machine-learning models compared to ensembles and human experts. Finally we discuss the threats to validity.

7.1 RQ1: What kind of pre-processing is required on engineering logs for usage in risk predicting machine-learning models?

Machine-learning models often require numerical, people tend to communicate through language rather than numerical values. Data related to the software engineering process therefore needs to be transformed. Some meta information relating to user stories can be categorically encoded, for example development platform. Other meta information might already be in a numerical format, for example how many people have worked on this story.
However the main content of engineering related logs is often text, which requires natural language processing. We used bag of words which counts occurrences of the words, creating a column for each word with the value being the amount of instances. This creates a large amount of columns, which can turn each row of data into an uniquely identifiable item for the machine-learning model. Which in turn can lead to the model overfitting on the data.
We compared this technique to "feature engineering" where other machine-learning models where used to extract information about the text. Each of the engineered features relates to a single column, giving the user control on the amount of columns added for text analysis. Overfitting becomes significantly less likely to occur.
We found that BoW can in some cases perform better, but its variance also seemed larger. The data set which was used also had a larger effect on the results when using BoW compared to feature engineering.
Finally all the Numerical data needs to be normalized as most ML algorithms have better performance with values in similar ranges.

7.2 RQ2: What kind of risk labels can be found in user stories?

Due to the nature of agile processes, data sets containing user stories labeled with risks are unlikely.
For some of the risks we researched we have found story level labels. However these labels are not a given for all software processes due to the variance between companies and even teams. Some of the labels we found also require pre-processing of related data before it can be available. We have created an overview of possible labels in table 7.1.
The risk label we used for our further experiments was story state, because it is usually present and can be used to infer the risk "focus on the wrong features". Which was deemed important by industry experts. However this is a rudimentary way of detecting this risk, since stories that are completed could have had a wrong focus. Our prediction system should help detect stories which might get cancelled, so they can be cancelled faster. However, using the label we chose won't necessarily find all stories that are unnecessary. Detecting which completed features, were unnecessary is a complex task, which we left out of scope for this project.

7.3 RQ3: How can machine learning based automation improve existing risk prevention in agile development processes?

To answer this question we first look at the sub-question:
RQ3.1: How do different ML algorithms behave when using user story data as input?

Risk	Story label
Focus on the wrong features	State
Scheduling too much work in a release	-
Source-code containing hard to manage technical debt	Technical debt score on changed code
Missing hard deadlines from external software providers (E.G. APIs end of life)	-
Modifying code with a lot of dependant code	Code call graph on changed code
Introducing bugs	Related pipeline test failures
Spending more time on tasks than estimated	Time worked
Missing knowledge to build a feature	Related "spike" stories
Losing members of the development team	-
Planning dependant tasks in the same increment	Linked stories

Table 7.1: Story level risk labels

The algorithms we chose had similar performance on the same data sets. We saw a larger amount of variance when processing the natural language in the data using bag of words. We attribute this to overfitting due to the very large column count. Out of the set of set of algorithms:

- Decision Tree
- Gaussian Naïve Bayes
- Multi-layer Perceptron
- Stochastic Gradient Descent
- Support Vector Machine
- k Nearest Neighbour

We found that GNB performed noticeably worse, this can be explained by the fact that our data does not fit this type of algorithm. Naïve Bayes algorithms work best when the data is not correlated to each other, however in our case the data was highly correlated. Because the story data was either coming from the same record, or was directly linked to it.
Depending on the data set, the models performed at around 65% accuracy or between 50% and 60% excluding the overfitted data. We have found that the data used to train the models for the risk predictions are at this stage more important than the specific algorithm which is used. As bigger improvements could be made by using data sets that are larger or have a better resolution.

To answer **RQ3**:
By including prediction results on user stories we can create a closer link between the risk and agile processes. To do this in an automated way so further work is needed to incorporate risk prediction in ITS systems. Our system could be used to create the prediction model, for such a system.

7.4 RQ4: What can be improved by applying ensemble ML techniques?

Across the different data sets and algorithms we see an improvement in **performance** using ensembles over regular models. The variance in ensemble algorithm seems higher than in the single algorithms. The result from the 4Industry data set with BoW text processing was mostly ignored due to the even more pronounced overfitting.
When looking at the differences in f1-score for singular model compared to the ensembles. We see that the f1-score for the "invalid" label improved going from 0.133 to 0.238. Ensembles show themselves to have a measurable effect on labels that are least presented in the data set. It therefore adds more value on data sets with a less even distribution. Noticeably the f1-score for human test subjects was distributed differently from the ML models and ensembles. They did worse on the most represented group compared to the single models and ensembles. However they did better on the data which was under represented in the set. The difference of score between "Done" and "Invalid" for people was 0.075, for the single models this difference was .684 and for the ensemble there was a .580 difference.

7.5 Threats to validity

In this section we will explain parts of the research that could affect the validity of the result. This is split into two parts, internal and external validity.

7.5.1 Internal validity

This section looks at possible problems relating to factors that are not influenced by outside factors.

Data inferences

Our system is composed of multiple parts where the results of each phase are not individually validated. Errors generated in the system are therefore harder to trace back to its origin, whether it be due to data error, a misconfiguration or something else.

Human expertise

The experiment using human experts to predict the same risk in stories was done with less data. As a machine can easily go through hundreds of stories, the experts time is constrained. The experts that validated the stories where not experts in the project. If experts involved with that project would have performed the estimations human performance could have turned out differently.

7.5.2 External validity

This section looks at issues with the generalizeability of the research.

Pipeline configuration

The configurations used for natural language processing in the experiments were not correlated with research about risk indicators. While we found that using the zero-shot analysis was beneficial in improving the prediction accuracy, we did not explicitly configure the zero-shot analysis to extract risk indicators from the text. However due to the complexity and time cost of validating the extraction of risk indicators from text, we left this out of scope and used the analysis to create general labels about the text.

Data set difference

Data sets have a large amount of effect on the output of machine learning algorithms. The results we achieved are tied to the data sets, the results between our different data sets seemed in-line with each other.
The chosen risk label is inferred and not manually labelled so it is possible that some of the data is incorrect. We did use two separate and unrelated data sets created by different groups of people. Which makes it unlikely institutionalized error is affecting all the results.
This all makes the specific results hard to generalize, however due to the configurability of the system results for another use case could easily be generated.

Chapter 8

Conclusion

In this thesis we explored risk assessment in agile software development. We wanted to know how risk assessment could be improved by using machine-learning techniques. To do this we needed to know how engineering logs from agile processes could be processed for the usage in ML algorithms. We required a way of labeling these engineering logs (mainly user stories) with risk information so the models could be trained. Thirdly we researched how agile processes could be improved using automated risk classification. Finally our focus went to ensemble algorithms to improve the results from singular machine-learning algorithms.
Software processes across the industry are varied between companies and teams. We could therefore not expect a fixed format for input data. To solve this we proposed a configurable data processing pipeline which prepares the data for the machine-learning algorithms.
A core part of these engineering logs comes in the form of natural language which needs to be processed. We proposed two methods of processing this, firstly using bag of words secondly feature pre-engineering (using existing ML models to create input columns). We found that BoW had a higher chance of causing overfitting in the machine-learning models.
To the best of our knowledge research has not yet found a specific algorithm which is ideal to use for the risk prediction use-case. To solve this the proposed system makes comparisons between multiple algorithms to achieve the best possible outcome. In doing this we found that there was not a specific algorithm which always seemed to perform better. The algorithms performed with similar accuracy, with the data being used having a larger effect on the end results.
We proposed the usage of ensemble methods to improve the prediction performance. We compared multiple ensemble algorithms, similarly to the singular algorithms we did not find a specific ensemble algorithm which was suited best for this task as it depended on the data. We did find that on average ensembles performed better than singular algorithms. Mainly because they had a higher prediction accuracy for the labels which where less represented in the data set. Compared to human experts both ensembles and singular algorithms performed better on data which was represented well in the data set. However experts had the edge when it came to data which was less occurring. For the Jira data set, human experts had an accuracy of 42% while the best ensemble had an accuracy of 68%.
By automating risk assessment the data can be integrated into the agile process without requiring sustained effort from experts. Research has found that agile processes can benefit from integrating risk management into a similar feedback loop as the Scrum sprint cycle. With automation a risk score can be added directly to the story making it transparent for the development team, allowing possible risks to be flagged early.

Chapter 9

Future work

Our research has identified gaps in the current literature and has tried to fill some of the gaps. We suggest the following topics to further research in this area.

Word2Vec
As an alternative to using bag of words or the feature pre-engineering. Using the "word2vec" algorithm [72] which learns proximity of words in relation to other words in the same corpus.
Reinforcement learning
This might be a way to improve the accuracy of these predictions, as it would allow the system to improve based on feedback. Such a feedback loop could also help with the challenge of generating risk labels for data.
Combining larger and smaller data scopes
This work looked at classifications by using stories as input, however by using data from whole sprints different classes of risk might become identifiable. When looking at a single story, it is hard to detect whether a there is a problem in the sprint. By looking on a level above stories, we hope that risk can be detected in relations between stories and their sprints.
Different risks
The scope of our research focused on a specific risk (focus on the wrong features), research could be performed looking at other risks in the same agile context. The challenge with this is creating or finding a labeled data set with risk on level of user stories. Similarly there is a part of the risk we investigated that could be investigated further. Features that where completed, but were "wrong" or not needed for the product. This again requires a labeled data set for this purpose.
Using risk indicators in text analysis
The models could be improved by further refining the input data, linking the text analysis to risk indicators before supplying them to the prediction model might further improve performance. However, this could also lead to larger biasses in the data which is pasted into the prediction model.
Pipeline performance improvements
The pipeline was built expecting algorithms to behave differently adding benefit to comparing them. However the performance of the algorithms is close and the comparisons and training of all these models adds a significant amount required computations. Research could be done on more data sets to find a single algorithm to lower computational cost.

Bibliography

[1] S. Y. Chadli and A. Idri, "Identifying and mitigating risks of software project management in global software development," *ACM International Conference Proceeding Series*, vol. Part F1319, pp. 12–22, 2017. DOI: 10.1145/3143434.3143453.

[2] M. Marinho, S. Sampaio, and H. Moura, "Managing uncertainty in software projects," *Innovations in Systems and Software Engineering*, vol. 14, no. 3, pp. 157–181, 2018, ISSN: 16145054. DOI: 10.1007/s11334-017-0297-y.

[3] P. Chandani and C. Gupta, "Requirement Risk Prioritization Using Analytic Hierarchy Process: A Gateway to Identify Risky Requirements," *2018 11th International Conference on Contemporary Computing, IC3 2018*, pp. 2–4, 2018. DOI: 10.1109/IC3.2018.8530569.

[4] J. McManus and T. Wood-Harper, "A study in project failure," *British Computer Society - The Chartered Institute for IT*, no. December, pp. 1–4, 2008. [Online]. Available: http://www.bcs.org/content/ConWebDoc/19584.

[5] PMI, "Success Rates Rise: Transforming the high cost of low performance," *Pulse of the Profession - 9th Global Project Management Survey*, pp. 1–32, 2017.

[6] M. Hammad and I. Inayat, "Integrating risk management in scrum framework," *Proceedings - 2018 International Conference on Frontiers of Information Technology, FIT 2018*, pp. 158–163, 2019. DOI: 10.1109/FIT.2018.00035.

[7] M. Niazi, S. Mahmood, M. Alshayeb, A. M. Qureshi, K. Faisal, and N. Cerpa, "Toward successful project management in global software development," *International Journal of Project Management*, vol. 34, no. 8, pp. 1553–1567, 2016, ISSN: 02637863. DOI: 10.1016/j.ijproman.2016.08.008. [Online]. Available: http://dx.doi.org/10.1016/j.ijproman.2016.08.008.

[8] T. Dingsøyr and Y. Petit, "4 Managing layers of risk: Uncertainty in large development programs combining agile software development and traditional project management," *Project Risk Management*, pp. 75–96, 2021. DOI: 10.1515/9783110652321-005.

[9] R. T. Nishijima, "THE CHALLENGE OF IMPLEMENTING SCRUM AGILE METHODOLOGY IN A TRADITIONAL DEVELOPMENT ENVIRONMENT José Gonçalo dos Santos Academic Discipline And Sub-Disciplines," vol. 5, no. 2, pp. 98–108, 2013.

[10] StateOfAgile, "14th annual STATE OF AGILE REPORT," *Annual Report for the STATE OF AGILE*, vol. 14, no. 14, pp. 2–19, 2020. [Online]. Available: https://digital.ai/catalyst-blog/the-14th-annual-state-of-agile-report.

[11] E. Talal Alharbi and M. R. Jameel Qureshi, "Implementation of Risk Management with SCRUM to Achieve CMMI Requirements," *International Journal of Computer Network and Information Security*, vol. 6, no. 11, pp. 20–25, Oct. 2014, ISSN: 20749090. DOI: 10.5815/ijcnis.2014.11.03. [Online]. Available: http://www.mecs-press.org/ijcnis/ijcnis-v6-n11/v6n11-3.html.

[12] B. G. Tavares, M. Keil, C. E. Sanches da Silva, and A. D. de Souza, "A Risk Management Tool for Agile Software Development," *Journal of Computer Information Systems*, vol. 00, no. 00, pp. 1–10, 2020, ISSN: 23802057. DOI: 10.1080/08874417.2020.1839813. [Online]. Available: https://doi.org/10.1080/08874417.2020.1839813.

[13] Ken Schwaber & Jeff Sutherland, "2020 Scrum Guide," no. November, 2020.

[14] J. Masso, F. J. Pino, C. Pardo, F. García, and M. Piattini, "Risk management in the software life cycle: A systematic literature review," *Computer Standards and Interfaces*, vol. 71, no. March, p. 103 431, 2020, ISSN: 09205489. DOI: 10.1016/j.csi.2020.103431. [Online]. Available: https://doi.org/10.1016/j.csi.2020.103431.

[15] W. W. Royce, "Managing the Development of Large Software Systems," *Ideas That Created the Future*, no. August, pp. 321–332, 1970. DOI: 10.7551/mitpress/12274.003.0035.

[16] R. Hoda, N. Salleh, and J. Grundy, "The Rise and Evolution of Agile Software Development," *IEEE Software*, vol. 35, no. 5, pp. 58–63, 2018, ISSN: 19374194. DOI: 10.1109/MS.2018.290111318.

[17] V. Mahnic, "Teaching scrum through team-project work: Students' perceptions and teacher's observations," *International Journal of Engineering Education*, vol. 26, no. 1, pp. 96–110, 2010, ISSN: 0949149X.

[18] J. Garzás and M. C. Paulk, "A case study of software process improvement with CMMI-DEV and Scrum in Spanish companies," *Journal of Software: Evolution and Process*, vol. 25, no. 12, pp. 1325–1333, Dec. 2013, ISSN: 20477473. DOI: 10.1002/smr.1605. [Online]. Available: http://doi.wiley.com/10.1002/smr.1605.

[19] B. G. Tavares, C. E. S. da Silva, and A. D. de Souza, "Risk management analysis in Scrum software projects," *International Transactions in Operational Research*, vol. 26, no. 5, pp. 1884–1905, 2017, ISSN: 14753995. DOI: 10.1111/itor.12401.

[20] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, 2019, ISSN: 19393520. DOI: 10.1109/TSE.2018.2792473.

[21] P. Pospieszny, B. Czarnacka-Chrobot, and A. Kobylinski, "An effective approach for software project effort and duration estimation with machine learning algorithms," *Journal of Systems and Software*, vol. 137, pp. 184–196, 2018, ISSN: 01641212. DOI: 10.1016/j.jss.2017.11.066. [Online]. Available: https://doi.org/10.1016/j.jss.2017.11.066.

[22] S. M. Satapathy and S. K. Rath, "Empirical assessment of machine learning models for agile software development effort estimation using story points," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 191–200, 2017, ISSN: 16145054. DOI: 10.1007/s11334-017-0288-z.

[23] Google, *Google Trends "devops"*, 2022. [Online]. Available: https://trends.google.com/trends/explore?date=2010-01-01%202022-01-04&q=devops.

[24] Wikipedia, the free encyclopedia, *Devops toolchain*, [Online; accessed April, 2022], 2022. [Online]. Available: https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg.

[25] F. M. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," *Journal of Software: Evolution and Process*, vol. 29, no. 6, pp. 1–20, 2017, ISSN: 20477481. DOI: 10.1002/smr.1885.

[26] T. Laukkarinen, K. Kuusinen, and T. Mikkonen, "Regulated software meets DevOps," *Information and Software Technology*, vol. 97, no. December 2017, pp. 176–178, 2018, ISSN: 09505849. DOI: 10.1016/j.infsof.2018.01.011. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.01.011.

[27] L. Karadsheh, S. AlHawari, E.-B. Naser, and M. W. Hadi, "Incorporating Knowledge Management and Risk Management as a single process," *Proceedings of the GBDI Tenth International Conference, Las Vegas, October*, 2008.

[28] X. Li, S. Liu, W. Cai, and S. Feng, "The application of risk matrix to software project risk management," *Proceedings - 2009 International Forum on Information Technology and Applications, IFITA 2009*, vol. 2, pp. 480–483, 2009. DOI: 10.1109/IFITA.2009.542.

[29] L. M. Alves, G. Souza, P. Ribeiro, and R. J. Machado, "Longevity of risks in software development projects: A comparative analysis with an academic environment," *Procedia Computer Science*, vol. 181, no. 2019, pp. 827–834, 2021, ISSN: 18770509. DOI: 10.1016/j.procs.2021.01.236. [Online]. Available: https://doi.org/10.1016/j.procs.2021.01.236.

[30] M. Pasha, G. Qaiser, and U. Pasha, "A critical analysis of software risk management techniques in large scale systems," *IEEE Access*, vol. 6, pp. 12 412–12 424, 2018, ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2805862.

[31] E. E. Odzaly, D. Greer, and P. Sage, "Software risk management barriers: An empirical study," *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, pp. 418–421, 2009. DOI: 10.1109/ESEM.2009.5316014.

[32] C. Haisjackl, M. Felderer, and R. Breu, "RisCal - A risk estimation tool for software engineering purposes," *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, pp. 292–299, 2013. DOI: 10.1109/SEAA.2013.10.

[33] G. Barish, M. Michelson, and S. Minton, "Mining commit log messages to identify risky code," *2017 World Congress in Computer Science, Computer Engineering and Applied Computing, CSCE 2017 - Proceedings of the 2017 International Conference on Artificial Intelligence, ICAI 2017*, pp. 345–349, 2017.

[34] M. N. Mahdi, M. Z. Mohamed, A. Yusof, L. K. Cheng, M. S. Mohd Azmi, and A. R. Ahmad, "Design and Development of Machine Learning Technique for Software Project Risk Assessment - A Review," *2020 8th International Conference on Information Technology and Multimedia, ICIMU 2020*, pp. 354–362, 2020. DOI: 10.1109/ICIMU49871.2020.9243459.

[35] M. Zavvar, A. Yavari, S. M. Mirhassannia, M. R. Nehi, A. Yanpi, and M. H. Zavvar, "Classification of risk in software development projects using support vector machine," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 9, no. 1, pp. 1–5, 2017, ISSN: 22898131.

[36] A. K. Sangaiah, O. W. Samuel, X. Li, M. Abdel-Basset, and H. Wang, "Towards an efficient risk assessment in software projects–Fuzzy reinforcement paradigm," *Computers and Electrical Engineering*, vol. 71, pp. 833–846, 2018, ISSN: 00457906. DOI: 10.1016/j.compeleceng.2017.07.022. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2017.07.022.

[37] H. R. Joseph, "Poster: Software Development Risk Management: Using Machine Learning for Generating Risk Prompts," *Proceedings - International Conference on Software Engineering*, vol. 2, pp. 833–834, 2015, ISSN: 02705257. DOI: 10.1109/ICSE.2015.271.

[38] E. E. Odzaly, D. Greer, and D. Stewart, "Agile risk management using software agents," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 3, pp. 823–841, 2018, ISSN: 18685145. DOI: 10.1007/s12652-017-0488-2.

[39] A. Albadarneh, I. Albadarneh, and A. Qusef, "Risk management in Agile software development: A comparative study," *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies, AEECT 2015*, 2015. DOI: 10.1109/AEECT.2015.7360573.

[40] R. J. Kusters, Y. Van De Leur, W. G. Rutten, and J. J. Trienekens, "When agile meets waterfall: Investigating risks & problems on the interface between agile & traditional software development in a hybrid development organization," *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, vol. 2, no. Iceis, pp. 271–278, 2017. DOI: 10.5220/0006292502710278.

[41] J. Nicolas, J. M. Carrillo De Gea, B. Nicolas, J. L. Fernandez-Aleman, and A. Toval, "On the Risks and Safeguards for Requirements Engineering in Global Software Development: Systematic Literature Review and Quantitative Assessment," *IEEE Access*, vol. 6, pp. 59 628–59 656, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2874096. [Online]. Available: https://ieeexplore.ieee.org/document/8482263/.

[42] M. Hammad, I. Inayat, and M. Zahid, "Risk management in agile software development: A survey," *Proceedings - 2019 International Conference on Frontiers of Information Technology, FIT 2019*, pp. 162–166, 2019. DOI: 10.1109/FIT47737.2019.00039.

[43] O. Ņikiforova, K. Babris, and J. Kristapsons, "Survey on Risk Classification in Agile Software Development Projects in Latvia," *Applied Computer Systems*, vol. 25, no. 2, pp. 105–116, 2020, ISSN: 2255-8691. DOI: 10.2478/acss-2020-0012.

[44] L. d. S. Leite, A. R. de Farias Neto, F. L. de Lima, and R. M. Chaim, "Analyzing and Modeling Critical Risks in Software Development Projects: A Study Based on RFMEA and Systems Dynamics," in *Trends and Applications in Information Systems and Technologies*, Á. Rocha, H. Adeli, G. Dzemyda, F. Moreira, and A. M. Ramalho Correia, Eds., Cham: Springer International Publishing, 2021, pp. 22–35, ISBN: 978-3-030-72654-6. DOI: 10.1007/978-3-030-72654-6{_}3.

[45] S. Beecham, T. Clear, R. Lal, and J. Noll, "Do scaling agile frameworks address global software development risks? An empirical study," *Journal of Systems and Software*, vol. 171, p. 110 823, 2021, ISSN: 01641212. DOI: 10.1016/j.jss.2020.110823. [Online]. Available: https://doi.org/10.1016/j.jss.2020.110823.

[46] Y. Hu, B. Feng, X. Mo, X. Zhang, E. W. Ngai, M. Fan, and M. Liu, "Cost-sensitive and ensemble-based prediction model for outsourced software project risk prediction," *Decision Support Systems*, vol. 72, pp. 11–23, 2015, ISSN: 01679236. DOI: 10.1016/j.dss.2015.02.003. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2015.02.003.

[47] J. S. Chou, M. Y. Cheng, Y. W. Wu, and C. C. Wu, "Forecasting enterprise resource planning software effort using evolutionary support vector machine inference model," *International Journal of Project Management*, vol. 30, no. 8, pp. 967–977, 2012, ISSN: 02637863. DOI: 10.1016/j.ijproman.2012.02.003. [Online]. Available: http://dx.doi.org/10.1016/j.ijproman.2012.02.003.

[48] A. B. Nassif, L. F. Capretz, D. Ho, and M. Azzeh, "A treeboost model for software effort estimation based on use case points," *Proceedings - 2012 11th International Conference on Machine Learning and Applications, ICMLA 2012*, vol. 2, pp. 314–319, 2012. DOI: 10.1109/ICMLA.2012.155.

[49] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving Cross-Project Defect Prediction Methods with Data Simplification," *Proceedings - 41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pp. 96–103, 2015. DOI: 10.1109/SEAA.2015.25.

[50] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," pp. 72–83, 2017. DOI: 10.1145/3106237.3106257.

[51] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-Aware just-in-Time defect prediction: Simple unsupervised models could be better than supervised models," *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, vol. 13-18-Nove, pp. 157–168, 2016. DOI: 10.1145/2950290.295035.

[52] E. d. S. Maldonado, E. Shihab, and N. Tsantalis, "Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, Nov. 2017, ISSN: 0098-5589. DOI: 10.1109/TSE.2017.2654244. [Online]. Available: http://ieeexplore.ieee.org/document/7820211/.

[53] J. Shivhare and S. K. Rath, "Software effort estimation using machine learning techniques," *ACM International Conference Proceeding Series*, 2014. DOI: 10.1145/2590748.2590767.

[54] Y. Hu, X. Zhang, X. Sun, M. Liu, and J. Du, "An intelligent model for software project risk prediction," *2009 International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2009*, vol. 1, pp. 629–632, 2009. DOI: 10.1109/ICIII.2009.157.

[55] V. S. Desai and R. Mohanty, "ANN-Cuckoo Optimization Technique to Predict Software Cost Estimation," *2018 Conference on Information and Communication Technology, CICT 2018*, 2018. DOI: 10.1109/INFOCOMTECH.2018.8722380.

[56] Y. Koroglu, A. Sen, D. Kutluay, A. Bayraktar, Y. Tosun, M. Cinar, and H. Kaya, "Defect prediction on a legacy industrial software: A case study on software with few defects," *Proceedings - International Conference on Software Engineering*, vol. 17-May-201, pp. 14–20, 2016, ISSN: 02705257. DOI: 10.1145/2896839.2896843.

[57] C. C. Kuo, C. L. Hou, and C. S. Yang, "The Study of a Risk Assessment System based on PageRank," *Journal of Internet Technology*, vol. 20, no. 7, pp. 2255–2264, 2019, ISSN: 20794029. DOI: 10.3966/160792642019122007022.

[58] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, *Agile Manifesto*, 2001. [Online]. Available: http://agilemanifesto.org/.

[59] K. E. van Oorschot, K. Sengupta, and L. N. Van Wassenhove, "Under Pressure: The Effects of Iteration Lengths on Agile Software Development Performance," *Project Management Journal*, vol. 49, no. 6, pp. 78–102, 2018, ISSN: 19389507. DOI: 10.1177/8756972818802714.

[60] T. Besker, H. Ghanbari, A. Martini, and J. Bosch, "The influence of Technical Debt on software developer morale," *Journal of Systems and Software*, vol. 167, 2020, ISSN: 01641212. DOI: 10.1016/j.jss.2020.110586.

[61] H. Al Hashimi and A. M. Gravell, "A Critical Review of the Use of Spikes in Agile Software Development," *Icsea 2019*, no. c, p. 166, 2019. [Online]. Available: http://eprints.soton.ac.uk/id/eprint/441801.

[62] Ken Schwaber & Jeff Sutherland, "2020 Scrum Guide," no. November, 2020.

[63] Z. Ma, R. Li, T. Li, R. Zhu, R. Jiang, J. Yang, M. Tang, and M. Zheng, "A data-driven risk measurement model of software developer turnover," *Soft Computing*, vol. 24, no. 2, pp. 825–842, 2020, ISSN: 14337479. DOI: 10.1007/s00500-019-04540-z. [Online]. Available: https://doi.org/10.1007/s00500-019-04540-z.

[64] Jira, *Configuring issue linking*, 2020. [Online]. Available: https://confluence.atlassian.com/adminjiraserver/configuring-issue-linking-938847862.html.

[65] J. Brownlee, *Data Preparation for Machine Learning*, 1.1, S. Martin, M. Sanderson, A. Koshy, A. Cheromskoy, and J. Halfyard, Eds. 2020. [Online]. Available: https://books.google.nl/books?hl=en&lr=&id=uAPuDwAAQBAJ&oi=fnd&pg=PP1&dq=how+to+prepare+input+for+machine+learning&ots=Cl2MzfaNqV&sig=UeAKSfCptK_e-SeVuM3_L0QztGE&redir_esc=y#v=onepage&q=how%20to%20prepare%20input%20for%20machine%20learning&f=false%20https://machinelear.

[66] K. Wiegers and J. Beatty, *Software Requirements 3rd Edition*. 2013, ISBN: 9780132344821.

[67] Github, *Github search (machine learning)*, [Online; accessed April, 2022], 2022. [Online]. Available: https://github.com/search?q=machine+learning.

[68] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," pp. 7871–7880, 2020. DOI: 10.18653/v1/2020.acl-main.703.

[69] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003, ISSN: 15324435. DOI: 10.1016/b978-0-12-411519-4.00006-9.

[70] J. Brownlee, *Data Preparation for Machine Learning*, 1.2, S. Martin, M. Sanderson, A. Kosh, A. Cheremskoy, and J. Halfyard, Eds. Machine Learning Mastery, 2020. [Online]. Available: https://books.google.nl/books?id=uAPuDwAAQBAJ.

[71] S. Bird, E. Klein, and E. Loper, *Natural language processing with python*, 1st. O'Reilly Media, Inc., 2009, p. 512, ISBN: 978-0-596-51649-9. [Online]. Available: https://dl.acm.org/doi/10.5555/1717171.

[72] K. W. CHURCH, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017. DOI: 10.1017/S1351324916000334.

Glossary

artifact A software artifact is something generated during the development process. These can be documents design or otherwise, executables, test scripts and more. 8

bag of words Is a technique used in natural language processing to represent text, where each word gets a counter for the number of occurrences. 1, 21, 24, 25, 28, 34, 35, 37, 38, 45

corpus A collection of written texts. 24

cyclomatic complexity Cyclomatic or McCabe complexity counts the control flow graph of a computer program. It measures the amount of paths that can be taken through the code. 10

DevOps DevOps is a set of practices and tools, that combine Dev (Development) and Ops (Operations), with a goal of shortening development life-cycles and provide continuous high quality deliveries. 7–9, 15

GIT A version control system, which tracks changes in a set of file, commonly used for code. 10

Tmux A terminal multiplexer which allows multiple session to be accessed simultaneously, it can also allow remote sessions to be active without the user having to be connected. 27

Acronyms

ANN Artificial Neural Network. 10, 12
API Application Programming interface. 14, 23, 24
BoW bag of words. 1, 21, 24, 25, 28, 31, 32, 34, 35, 37, 38
DAD Disciplined Agile Delivery. 11
DT Decision Tree. 12, 22, 30–32, 35
fmGA Fast Messy Genetic Algorithm. 12
GNB Gaussian Naïve Bayes. 28, 35
GNB Multinominal Naïve Bayes. 28
ITS Issue Tracking System. 1, 17–19, 24
k-NN k Nearest Neighbour. 12, 28, 31, 35
LSTM Long Short-term Memory. 12
ML machine-learning. 1, 6–8, 10, 12, 17–28, 30, 34, 35, 37
MlP Multi-layer Perceptron. 22, 28, 31, 35
NB Naïve Bayes. 12, 22, 35
NLP Natural Language Processing. 1, 2, 14, 21
PO Product Owner. 18, 48
REST Representational State Transfer. 23
RF Random Forest. 12
RHN Recurrent Highway Network. 12
SAFe Scaled Agile Framework. 11
SDLC Systems Development Life Cycle. 7, 8
SGD Stochastic Gradient Descent. 35
SVM Support Vector Machine. 10, 12, 35

Appendix A

Risk information

A.1 Engineering logs

Label	Value
Feature	feature
Defect	defect
Research Spike	research_spike
Architecture	architecture
Security	security
Technical Debt	technical_debt

Table A.1: Story classification

Label	Value
Critical	1
High	2
Moderate	3
Low	4
Planning	5

Table A.2: Story priority

Field	Type	Format
acceptance_criteria	String	html formatted
blocked_reason	String	
closed_at	String	Date (YYYY-MM-DD HH:mm:ss)
comments_and_work_notes	String	
detailed_description	String	html formatted
number	String	ASTRY00001
opened_at	String	Date (YYYY-MM-DD HH:mm:ss)
proposed_solution	String	html formatted
short_description	String	
sys_created_by	String	email of user
sys_created_on	String	Date (YYYY-MM-DD HH:mm:ss)
sys_id	String	uuid
sys_updated_by	String	email of user
sys_updated_on	String	Date (YYYY-MM-DD HH:mm:ss)
work_notes	String	
assigned_to	Reference	uuid, record name
assignment_group	Reference	uuid, record name
closed_by	Reference	uuid, record name
epic	Reference	uuid, record name
feature	Reference	uuid, record name
opened_by	Reference	uuid, record name
release	Reference	uuid, record name
sprint	Reference	uuid, record name
theme	Reference	uuid, record name
reassignment_count	Number	
sys_mod_count	Number	
points	Integer	
classification	Choice(String)	
priority	Choice(Number)	Choice name, number
state	Choice(Number)	Choice name, number
active	Boolean	
blocked	Boolean	

Table A.3: User story data

A.2 Survey information

ID	Description
RSK001	Does the story contain ambiguity that can be misinterpreted?
RSK002	Does the story contain any conflicting information?
RSK003	Not being able to release a version of the product timely (missing planned time to market)
RSK004	Allowing technical debt to become to large
RSK005	Focus on the wrong features
RSK006	Misinterpret customer wishes to build something they don't want
RSK007	Not knowing the product well enough when introducing a new feature
RSK008	Planning to much work into a release
RSK009	Misunderstanding the PO's intent of the story
RSK010	Lacking technical knowledge to build a feature
RSK011	Not taking non-functional requirements into account, leading to possible insecure, slow or other problems
RSK012	Dev not properly checking their code works / not testing
RSK013	Code being changed has impact on large part of the system
RSK014	Code being changed is unstable and changes can easily cause bugs
RSK015	Development taking longer on a task than expected due to unforeseen complexity
RSK016	High performing team members / knowledge leaving the team
RSK017	Planning work that is dependent on other work which is still planned / in progress
RSK018	Not reviewing code strict enough allowing bad code to get merged
RSK019	Having a slow review process, causing constant merges to be required that might create merge conflict bugs
RSK020	Misinterpreting the acceptance criteria sending the story back to be reviewed by developers
RSK021	Not testing enough
RSK022	Adding technical debt deliberately
RSK023	Hard deadlines from external software providers
RSK024	Accidentally introducing bugs
RSK025	Working on code that has a lot of dependent code
RSK026	Introducing bugs in code that has a lot of dependants
RSK027	Not finding bugs before shipping the product

Table A.4: Initial risks

Appendix B

Code examples

The full code base for the pipeline can be found on github at the following address: `https://github.com/DriesMeerman/Risk-Assessment`.

```
from sklearn.preprocessing import MinMaxScaler
from numpy import asarray

scaler = MinMaxScaler()

data = asarray([
    [0, 0],
    [2, 500],
    [8, 1000],
    [7, 800]
])

res = scaler.fit_transform(data)
display(res)
# array([[0.    , 0.    ],
#        [0.25  , 0.5   ],
#        [1.    , 1.    ],
#        [0.875, 0.8   ]])
```

Listing B.1: MinMax Scaler example

```
#!/bin/bash
echo "project,hash,author,email,isodate,subject,body" > ./commit_info.csv
for d in */ ; do
    cd $d;
    result=${PWD##*/};
    git log --pretty="format:%h,%cn,%ce,%cI,%s,\"%b\"<;>" | tr '\n' '␣' | sed 's/<;>/\n/g' |
        awk -v project=$result '{printf"%s,%s\n",␣project,␣$0}' > ../projectInfos.csv;
    cd ..;
done;
```

Listing B.2: Commit scraper

Depending on your system the "#!/bin/bash" shebang might need to be adjusted. Due to the fact that some commit messages where using string character such as ' and " a more thorough version had to be created. But if the commit messages are known not to contain such characters, the initial version of the script should be used as it runs seconds rather than minutes in our case for the same dataset of 11988 commits. This is due to the second script doing IO operations for each commit rather than for each repository.

```
#!/bin/bash
function escape_chars {
    sed -e 's/"/\\"/g' -e "s/'/\\\\'/g"
}

function trim {
    awk '{$1=$1;print}'
}

function format {
    sha=$(git log -n1 --pretty=format:%H $1 | trim)
    author=$(git log -n1 --pretty=format:%cn $1 | escape_chars)
    email=$(git log -n1 --pretty=format:%ce $1 | escape_chars)
    date=$(git log -n1 --pretty=format:%cI $1)
    subject=$(git log -n1 --pretty=format:%s $1 | escape_chars)
    body=$(git log -n1 --pretty=format:%b $1 | tr '\n' '␣' | escape_chars)
    project=$2

    echo "$project,$sha,$author,$email,$date,\"$subject\",\"$body\""
```

```
}

function get_repo_logs {
    for hash in $(git rev-list --all); do
        format $hash $1
    done
}

echo "Started: $(date)\n"
SECONDS=0

filename="commit_invo.csv"
echo "project,hash,author,email,isodate,subject,body" > $filename;

for project_folder in */ ; do
    cd $project_folder;
    echo "Working on $project_folder"
    project_name=${PWD##*/};
    get_repo_logs $project_name >> ../$filename;
    cd ..;
    duration=$SECONDS
    echo "$(($duration / 60)) minutes and $(($duration % 60)) seconds elapsed."
done
echo "Finished: $(date)\n"
```

Listing B.3: Commit scraper with escaping

Appendix C

Data schema's

Below we show the json structure of a story retrieved from Jira, some fields have been omitted for brevity. Depending on the configuration of the Jira host the custom fields can be different.

```
{
    "id": "447554",
    "self": "string(issue_link)",
    "key": "string(id)",
    "fields": {
        "summary": "string",
        "description": "string",
        "status": "string",
        "type": "string",
        "labels": "string",
        "project": "string",
        "assignee": "string",
        "created": "date",
        "updated": "date",
        "resolutiondate": "date(resolutiondate)",
        "customfield_10202": "number(points)",
        "customfield_15602": "string(urgent)",
        "customfield_12906": "string(severity)",
        "customfield_12908": "string(probability)",
        "customfield_12905": "string(probability_final)",
        "customfield_12909": "string(harm)",
        "customfield_16701": "string(risk_likelyhood)",
        "customfield_16700": "string(risk_impact)",
        "customfield_15204": "string(requirement_prio)",
        "customfield_10205": "string(sprint)"
    }
}
```

Listing C.1: Jira Story JSON schema