An Analysis of the State of the Art of Machine Learning for Risk Assessment in Software Projects

André Sousa[1], João Pascoal Faria[2], João Mendes-Moreira[2]
[1]*Master's in Software Engineering,* [2]*Departament of Informatics Engineering*
Faculty of Engineering of the University of Porto
Porto, Portugal
{up201902618, jpf, jmoreira}@fe.up.pt

***Abstract*—Risk management is one of the ten knowledge areas discussed in the Project Management Body of Knowledge (PMBOK), which serves as a guide that should be followed to increase the chances of project success. The popularity of research regarding the application of risk management in software projects has been consistently growing in recent years, particularly with the application of machine learning techniques to help identify risk levels or risk factors of a project before the project development begins, with the intent of improving the likelihood of success of software projects.**

This paper provides an overview of various concepts related to risk and risk management in software projects, including traditional techniques used to identify and control risks in software projects, as well as machine learning techniques and methods which have been applied to provide better estimates and classification of the risk levels and risk factors that can be encountered during the development of a software project. The paper also presents an analysis of machine learning oriented risk management studies and experiments found in the literature as a way of identifying the type of inputs and outputs, as well as frequent algorithms used in this research area.

***Index Terms*—Risk Management, Risk Assessment, Software Projects, Machine Learning, Classification**

I. Introduction

According to the Project Management Body of Knowledge, a project risk is "an uncertain event which, if it occurs, has a positive or negative effect on one or more project objectives" [1]. Software projects are notoriously complex development activities, and thus the concept of risk cannot be ignored when considering this type of projects.

In 2015, the Standish Group International's CHAOS Report [2], a study of the success of software projects, reported a 29% success rate for the roughly 5000 projects investigated. A project is considered successful if it is completed within its allocated budget, original delivery deadline, and with all of the features that were planned at the start of its development life cycle [3]. It also reported a 19% failure rate for the set of projects investigated, meaning the projects suffered from cost or time overruns, or lacked content that was initially specified.

However, it is in the category of challenged projects that we can find the largest percentage of projects. From 2011 to 2015, 49%, 56%, 50%, 55%, and 52% of the software projects, respectively, were considered challenged, meaning they were completed but either over-budget, over the allocated time estimates, or offering fewer features than originally planned. The consistently high percentage of challenged projects indicates that there is room for improvement in the success rate of a large amount of these projects. This is where the concepts of risk and risk management are important to consider.

Risk management is a process used for early identification, analysis, planning, and control of risks in a project [4], with the goal of minimizing negative risks and maximizing positive risks [5], also referred to as opportunities. By identifying the risks and creating mitigation plans to deal with them if they occur before the project development starts rather than coming up with strategies to deal with risks in the moment they materialize, project managers and development teams are better prepared to handle risks and their effects on a project, which in turn can lead to more projects being completed on time and within their allocated budgets. However, it is typically the first activity to be removed from the project management activities when a project falls behind schedule [6].

In recent years, there has been an increasing use of machine learning algorithms and techniques for risk assessment, particularly supervised learning ones where the model is trained using a data set, and then the same model is used to predict information on a new set of data (in this area, it could be to predict possible risk factors of a project based on its characteristics, such as team members, time, and allocated budget). Commonly used algorithms for supervised learning include Decision Trees, Naive Bayes classifiers (NB), Neural Networks (NN), and Support Vector Machines (SVM).

The purpose of this paper is to provide an overview of the state-of-the-art in topics related to risk and risk management with regards to their application in the management of software projects, including traditional processes as well as the growing application of machine learning techniques to tackle the problems associated with risk assessment in software projects. The remaining sections of the paper are structured as follows. An overview of the concepts of risk and risk management in software projects, and different types of risks and risk management processes are presented in section 2. In section 3, a literature review of the application of machine learning techniques for risk management is performed, showing examples of their use in the prediction of risk levels of software projects. Lastly, section 4 concludes the paper.

DOI reference number: 10.18293/SEKE2021-097

II. Risk and Risk Management in Software Projects

Risk in software projects can be seen as "the potential that a chosen action or activity will lead to a loss or an undesirable outcome" [5]; "a set of factors or conditions that can pose a serious threat to the successful completion of a software project" [7]; or "the probability and impact of an event on a project" [8]. From these definitions, it is possible to identify some common themes, such as a risk possibly leading to a loss. In a software project, a loss can manifest itself through lower quality of the final product, increased costs, changes to the release date of the product, or, in a worst-case scenario, failure and cancellation [9].

Software projects can be impacted by various types of risks [10]:

- **Technical risks** - problems with the programming languages and frameworks of choice, project size, or processes. This type of risk can occur as a result of lack of experience or lack of maturity of the technologies used.
- **Management risks** - these risks can occur due to problems in communication with top management and customers, lack of planning, or lack of project management experience.
- **Financial risks** - problems regarding budget, cash flow, or doubts about the return on investment of the project.
- **Contractual and legal risks** - problems regarding adjusting the schedule or the requirements to fit the market, government regulations, or health and safety problems.
- **Personnel risks** - these can be due to conflicts among staff, ethical and moral issues, or productivity issues resulting from a combination of the aforementioned risks.
- **Other resource risks** - these occur due to situations that are generally not a responsibility of the project team, such as unavailability of computer resources or equipment.

When it comes to the most frequent specific risks in software projects, Boehm, regarded by many as one of the most important authors in this research area, listed risks such as personnel shortfalls, unrealistic schedules and budgets, and developing the wrong functions and/or user interface as some of the most frequent risks that have a direct effect on the success of software projects [8].

The high percentage of challenged projects seen in the Standish Group International's CHAOS reports [2] is consistent with the information presented by Boehm [8], as the first and second most frequent risk items listed (personnel shortfalls and unrealistic schedules and budgets) are directly related to the concept of challenged projects, and often come as a consequence of the majority of the remaining risk items listed occurring during the development of a project.

To reduce the high percentage of challenged projects in the software industry, project managers must consider a wide variety of knowledge areas in order to manage their projects towards successful completion. One of those areas is risk management, as risks can be identified in various areas of a software project. In a software development project, risks can be influenced by the business domain, the business style, culture of the organization, and characteristics of the members involved in the project [11], so it is important to identify risks according to the environment in which the project is being developed. To facilitate this process, risk factor and item classifications found in the literature can be used. These classifications usually list the most frequent risk items that can affect a project's path towards success, and teams can use these to evaluate if there is a possibility of any of those risks occurring during the development of their own projects, and if so, what could be their impact on the project. Essentially, those are the 2 parts that make up a risk: the likelihood of the risk happening, and the degree of impact it has on the project if it does occur.

Wallace's categorization [7] of risk items according to six risk dimensions (Team, Organizational Environment, Requirements, Planning and Control, User, and Complexity) is still widely used in this research area. Not only does it present common risk items in software projects, but by grouping them according to a specific dimension within the areas that project managers have to consider in the development of a software project, it makes it so they can identify what areas are more likely to be problematic throughout the course of the development of the project and prepare their risk management strategies accordingly.

Those are just some examples of risk classifications that can negatively affect specific areas during the development of a software project. In reality, there are a lot more risks that can be identified, and doing it at an early stage of the project (ideally before development starts) is crucial for a successful development life cycle, as it means the project manager as well as the development team can start to plan actions to take if these risks materialize during the project development.

A good way of classifying the identified risks according to their priority is through the use of a portfolio chart, such as the one presented by Dr. Ernest Wallmüller [12], which can be seen in table I.

TABLE I
Risk priorities according to their probability and impact

Impact				
High	B	A	A	
Medium	C	B	A	
Low	C	C	B	
	Low	Medium	High	**Probability**

Identifying, classifying, and prioritizing actions for risks according to their priority are just some of the phases of a process called risk management, which is defined by Standard ISO/IEC/IEEE 24765:2010 as "an organized process for" identifying and handling risk factors; assessing and quantifying the identified risks; and developing plans to deal with the identified risks [13].

In a very simplified way, the goal of risk management is to increase the probability of positive events on a software project, while at the same time decreasing the probability of negative events on the same project [4]. To do that, risk

management is often divided into two key activities: risk assessment and risk control, which are composed of more specific steps.

In [8], Boehm split risk assessment into three phases - identification, analysis, and prioritization - and risk control into three other phases: management planning, resolution (or mitigation), and monitoring. Other researchers may change the categories to which these steps belong to, such as in [14], where risk assessment is made up of phases for identification, analysis, prioritization, planning, and resolution, and risk control is made up of only one phase, which is monitoring, but the key concepts and phases remain the same.

Risk assessment begins with the risk identification phase, where a list of risk items related to the project that have a higher chance of affecting the success of the project is created. Common techniques in this phase include checklists and decision-driven analysis. Afterwards, the loss probability and impact of each identified risk item is assessed in the risk analysis phase. Analysis of factors such as quality, reliability, and availability is a common task in this phase. However, there is usually some uncertainty when it comes to estimating the losses that are a result of the occurrence of a risk [8], so the assessments done are very often subjective, and often the result of interviewing domain experts [8]. Next, in the risk prioritization phase, the identified risks are ordered through the use of techniques such as the analysis of risk exposure and risk reduction.

With the first three phases of risk management (according to Boehm) completed, the risk control activities can begin with the risk management planning step, which addresses the risk items identified through processes such as buying information (e.g., investing in a prototype to better understand the specific risk), risk avoidance, risk reduction, risk transfer, and risk plan integration. Common techniques used in this step are checklists of risk-resolution techniques, cost-benefit analysis, and risk management plan outlines. Afterwards, in the risk resolution phase, the identified risk items are analyzed and decisions are taken regarding what action to take against the risks in order to mitigate them. Boehm identified several fundamental risk mitigation strategies, such as understanding the risk or removing the risk from the project's critical path [4].

Lastly, in the risk monitoring phase, the project's progress is tracked towards completion by resolving the previously identified risk items and taking corrective action whenever necessary through the use of techniques such as milestone tracking and risk assessment.

Boehm's risk management model is frequently referenced in the literature, but there are several other traditional risk management models and processes that can be found in the literature. The authors in [15] analyzed several risk management models and processes, such as:

- **Team Risk Management (TRM)** [16] - risks are managed in the full software development life cycle, and all members and stakeholders are involved, improving the efficiency of the decision-making process. TRM frequently ensures continuous risk management through regular reviews and monitoring of the implemented processes.
- **Softrisk management technique** [17] - this technique is constructed on the basis of documentation and gives special focus to extreme risks by focusing on what can be leading to those risks. Re-estimation, re-prioritization, re-assessment, and re-documentation are performed to also guarantee continuous risk management.
- **Wallmüller's Risk Management Process** [12] - risk management activities are conducted by the project team at the same points where the cost, time, quality, and requirement management activities are performed. A major point of difference compared to the previous models is the introduction of risk management roles which are assigned to different members of the team, thus making sure the entire team contributes to the risk management tasks and is up-to-date on the status of risks in the project.

There is another area that has been gaining a tremendous amount of attention, particularly in recent years, with the goal of improving risk management processes in software projects, and that is the application of machine learning techniques and methods to improve the risk management workflow in software development companies.

III. Literature Review of Machine Learning Approaches in Risk Management

Machine learning in risk management has obtained increasing popularity in recent years, and a lot of different approaches have been used. For the purposes of the analysis of the state of the art performed in this paper, the focus was on finding practical applications of machine learning to predict possible project risks or an overall risk level of a project. From there, it was possible to identify not only some of the most frequently used algorithms and evaluation metrics, but also the type of information used as inputs used to train the models.

Throughout the creation of this paper, bibliographic databases such as Scopus and DBLP were used to search for various articles, scientific papers, and surveys related to this topic. Searches were performed using keywords such as "risk assessment", "machine learning", and lastly "software projects" to reduce the scope of the results to the application of machine learning for risk assessment specifically in the software development industry. By reading the abstracts and briefly analysing the contents of the search results, the ones that were considered more relevant were read and analysed in more detail. Some examples of studies and experiments done in this research area are described below, and can be seen in greater detail in terms of inputs and outputs used in table II.

In [18], an Artificial Neural Network model was created to predict deviations in new software projects. The inputs to the model were the risk factors detected in the projects, and the outputs were the differences found in time, budget, and number of personnel, number of completed work packages, and success of the project under investigation. This experiment showed the applicability of Neural Networks when the intended information spans more than one category (in this

case, the deviations in five attributes related to the project), as well as the fact that the model can have a great performance and accuracy, as seen in its results.

A Neural Network model was also created in [19], together with a Support Vector Machine model to compare both approaches and their accuracy in evaluating the risk level of software projects. The input used was a vector of risk factors of 120 software projects, collected after several interviews with experts in the industry, which were then grouped according to six different risk categories (Environment Complexity, Project Requirement Complexity, Cooperation, Team, Project Management, and Engineering). The output was the predicted outcome of the project ("successful", "failed", or "challenged"). The Support Vector Machine model had a higher accuracy compared to the Neural Network method (80% vs %70, respectively) due to NN's tendency in finding a local optima [19], but after changes were made to the NN method by optimizing it with a Genetic Algorithm (GA), this made it so the NN-GA method surpassed SVM in accuracy (85% vs 80%, respectively) by reducing the search for a local optima.

In [20], the author proposes a Neural Network architecture with a back propagation algorithm to learn the patterns of a data set of projects completed in the past, which also includes 22 project risk factors of areas such as estimations, requirements (e.g., frequent changes to requirements), and team organization (e.g., lack of skills or experience). The output of the model was a classification of the risk level of the project: "risky" or "not risky". The model developed was found to have a higher accuracy and sensitivity when compared to a Logistic Regression model developed from and applied on the same data set.

The authors in [11] developed an approach to predict runaway projects (projects that greatly exceed budget and deadlines and have failed to produce an acceptable deliverable) in an organization through the use of a questionnaire to identify the characteristics of projects, and then classify them into "runaway" or "success" projects through the use of a Naive Bayes classifier. These characteristics are classified according to five different categories: requirements (e.g., ambiguity of requirements), estimations (e.g., lack of stakeholders present for estimation process), planning (e.g., unspecified milestones), team organization (e.g., lack of skills or experience), and project management (e.g., inadequate project monitoring). 10-fold cross validation was used to evaluate the effectiveness of their solution, showing a predictive accuracy of 82.5%, with 33 out of 40 projects classified correctly.

Bayesian classifiers were also used in [21] and [22]. In the former, a Bayesian Belief Network (BBN) was used to build a software risk estimation model that was used for the main software risk indicators for risk assessment in software projects. In the latter, a model was created using a Bayesian network with causality constraints to identify and analyze risks in software development projects through data collected from 302 software projects. The authors found that it had an accuracy of 1% to 7% higher than the other models tested (Logistic Regression, Decision Tree, and Naive Bayes), which they attributed to the incorporation of expert domain knowledge and causality discovery into the BBN.

In [23], the authors used a Support Vector Machine to model risk classification in software projects. The model classified projects as either high risk or low risk. SVM was also used in [24] to predict the risk level of different projects as either "low", "medium", or "high". A Neural Network was used for comparison, and the authors found that the SVM was more accurate (85% accuracy of SVM compared to 75% of the NN).

Multiple Logistic Regression was used in [25] to classify different characteristics of software projects as either a "risk" or a "non risk". The input data was obtained through questionnaires sent to experts in the software project development and management fields, which asked them to classify risk factors from 8 categories (User, Requirements, Estimations, Cost, Schedule, Planning and Control, Team, Software) according to their risk level on a scale from 1 to 5.

Lastly, the authors in [26] used Logistic Regression to classify projects as either "risky" or "not risky". Responses to a questionnaire focusing on 5 viewpoints of key risk factors (Requirements, Estimations, Planning, Organization, Management) were used as the input data, and the model developed classified 35 out of 40 projects correctly.

As can be seen, there are a lot of possibilities when it comes to machine learning models that can be used to predict risks in software projects. However, there are definitely areas in this field that can be explored further in order to improve the applicability of machine learning models for risk assessment.

Some of the papers presented in table II compare different machine learning algorithms (e.g., [19] and [24]) with the goal of comparing their predictive performance in the context of a specific problem. However, a greater focus should be placed in also comparing them in terms of interpretability and the performance trade-offs involved in more interpretable algorithms.

Interpretability in machine learning is defined as "the degree to which a human can understand the cause of a decision" [27]. Interpretable machine learning models make it easier to understand not only the prediction made by the model, but more importantly why that prediction was made. If a prediction does not match what was initially expected, developers can use this information to identify possible issues in the data set, the model, or possibly both. However, there is a trade-off involved with interpretable machine learning algorithms, namely the fact that predictive performance tends to be lower with these algorithms.

Additionally, considering the popularity of project management software such as JIRA and Asana, one of the next areas of focus in this research field could be the creation of machine learning models that can be integrated with these tools. This integration with tools used for daily project management tasks could make it so risk management becomes just another step in the project management cycle, rather than a process which requires a large overhaul in an organization's workflow in order to integrate it in their processes.

TABLE II
STUDIES AND EXPERIMENTS ON THE USE OF MACHINE LEARNING TECHNIQUES FOR RISK ASSESSMENT

Reference	Inputs	Outputs	Algorithm(s)	Evaluation metric(s)
A Novel Model for Risk Estimation in Software Projects using Artificial Neural Network [18]	45 risk factors of 20 software projects (70% of data used for training, 30% for testing)	Deviations in project duration, cost, number of personnel, completed work packages, project success	Neural Network	Training R[1]= 0.9978 Testing R = 0.9935 Validation R = 0.996 MSE[2] = 0.001
Software Project Risk Management Modelling with Neural Network and Support Vector Machine Approaches [19]	Data of 120 software projects collected through questionnaires distributed in cities in China (83.3% of data used for training, 16.7% for testing)	Classification of projects as either "successful", "challenged", or "failed"	Neural Network	Accuracy = 70%
			Genetic Algorithm NN	Accuracy = 85%
			Support Vector Machine	Accuracy = 80%
Discriminating Risky Software Project Using Neural Networks [20]	22 attributes of 40 projects in the OMRON database (80% of data used for training, 20% for testing)	Risk level of the project - "risky" or "not risky"	Neural Network	Accuracy = 82.2% Precision = 81.82% TPR[3] = 81.82% TNR[4] = 82.61%
			Logistic Regression	Accuracy = 87.5% Precision = 100% TPR = 66.7% TNR = 100%
An Empirical Evaluation of Predicting Runaway Software Projects Using Bayesian Classification [11]	Responses on a 4 point Likert scale to a questionnaire focusing on 5 viewpoints of key risk factors in 40 SSBC projects (10-fold cross-validation used for testing)	Project classification as either "runaway" or "success"	Bayesian classifiers	Accuracy = 82.5%
A Probabilistic Software Risk Assessment and Estimation Model for Software Projects [21]	Assessment of 27 risk factors (low, medium or high) in 12 software projects	Probability of the project being of low, medium, or high risk	Bayesian classifiers	MMRE[5] = 0.03842 BMMRE[6] = 0.03911
Software Project Risk Analysis using Bayesian Networks with Causality Constraints [22]	Software project data from 302 projects collected through questionnaires (10-fold cross-validation used for testing)	Classification of project's performance based on risks identified as "low" or "high"	Bayesian network with causality constraints	Accuracy = 75.15%
			Decision Trees	Accuracy = 70.86%
			Naive Bayes	Accuracy = 72.85%
			Bayesian classifiers	Accuracy = 74.17%
Classification of Risk in Software Development Projects using Support Vector Machine [23]	**530 samples of a data set created from information of software development projects (70% of data used for training and 30% for testing)**	**Project risk classification as either "low risk" or "high risk"**	**Support Vector Machine**	**Accuracy = 99.51% AUC[7] = 98%**
An Intelligent Model for Software Project Risk Prediction [24]	64 risk factors of data from 120 projects (83.3% of data used for training, 16.7% used for testing)	Classification of projects as either "successful", "challenged", or "failure"	Neural Network	Accuracy = 75%
			Support Vector Machine	Accuracy = 85%
Prediction of Risk Factors of Software Development Project by Using Multiple Logistic Regression [25]	**Data obtained from questionnaires regarding the risk level of 70 software projects**	**Classification of characteristics of a software project as "risk" or "non risk"**	**Multiple Logistic Regression**	**Accuracy = 90%**
An Empirical Approach to Characterizing Risky Software Projects Based on Logistic Regression Analysis [26]	**Responses on a 4 point Likert scale to a questionnaire focusing on 5 viewpoints of key risk factors in 40 SSBC projects**	**Classification of projects as either "risky" or "not risky"**	**Logistic Regression**	**Accuracy = 87.5%**

[1] Defined by the authors as Regression value, indicating the correlation between the predicted values and the observed values. A higher value indicates better results.
[2] Mean Squared Error. A smaller value indicates better results.
[3] True Positive Rate - the percentage of positive cases that were correctly identified. A higher value indicates better results.
[4] True Negative Rate - the percentage of negative cases that were correctly classified. A higher value indicates better results.
[5] Mean Magnitude of Relative Error. A lower value indicates better predictive performance.
[6] Balanced Mean Magnitude of Relative Error. Due to the fact that MMRE penalizes overestimates more than underestimates, a balanced MMRE is also used in this experiment. As with MMRE, a lower value indicates better predictive performance.
[7] Area Under the ROC Curve - the probability of the model ranking a random positive example higher than a random negative one. AUC returns a value between 0 and 1, where the higher the AUC, the better the model is at distinguishing positive and negative classes.

Lastly, the use of accuracy as the sole evaluation metric to assess the quality of the models developed (e.g., [19], [25], [26]) is sometimes not enough. The usefulness of accuracy as an evaluation metric depends on the balance of the classes in the problem at hand. As an example, if there are three possible risk levels (e.g., low, medium, and high) to choose from in the data set's dependent variable, and 70% of the samples are of class "low", 10% are of class "medium", and the remaining 20% are of class "high", the model can easily obtain a high training accuracy by just predicting the majority of the testing samples to be of class "low". In classification problems, additional metrics such as AUC, True Positive Rate, and True Negative Rate should also be closely looked at to determine if the model is truly returning good results, or if it is only achieving a high accuracy by predicting the testing samples as being of the majority class most of the time.

IV. Conclusion

As software projects can face a lot of different problems before they are released to the market, it is important to at least identify possible risks that can occur before development starts, making it possible to start planning risk management and mitigation strategies if the risks materialize, rather than dealing with the problems as they appear. Risk management in software projects is a research area with consistently growing popularity, especially when combined with machine learning approaches to create models that can identify or predict risks before project development starts, with the goal of identifying risks in a software project, and ultimately develop and implement strategies to prevent or limit the impact of the identified risks if they materialize during the project's development.

Explainable AI is also a research field with increasing research that should be considered to explain the prediction of black-box models, such as Neural Networks or Support Vector Machines. One of the next steps in this research area should be to focus on understanding the predictions that are made by the models by using interpretable models or black-box models but, in the latter case, with their predictions explained by explainable AI. Which would be the best?

Lastly, the creation of machine learning models in software packages that can then be integrated with popular project management tools such as JIRA or Asana should be analysed more closely.

References

[1] PMI, *A Guide to the Project Management Body of Knowledge (PMBOK Guide), 4th Edition.* Project Management Institute, 2008.

[2] T. S. Group, "Chaos report 2015," 2015. [Online]. Available: https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

[3] M.-Y. Hsieh, Y.-C. Hsu, and C.-T. Lin, "Risk assessment in new software development projects at the front end: a fuzzy logic approach," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, 04 2016.

[4] B. Boehm, "Software project risk and opportunity management," *Software Project Management in a Changing World*, pp. 107–121, 03 2014.

[5] P. Chawan, J. Patil, and R. Naik, "Software risk management," *International Journal Of Computers & Technology*, vol. 6, pp. 60–66, 05 2013.

[6] Y. Kwak and J. Stoddard, "Project risk management: Lessons learned from software development environment," *Technovation*, vol. 24, pp. 915–920, 11 2004.

[7] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," *Information & Management*, vol. 42, no. 1, pp. 115 – 125, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378720604000102

[8] B. Boehm, "Software risk management: principles and practices," *IEEE Software*, vol. 8, pp. 32–41, 1991.

[9] R. C. Williams, G. J. Pandelios, and S. Behrens, "Software risk evaluation (sre) method description (version 2.0)," 2000.

[10] L. Westfall, "Defining software risk management," 2001. [Online]. Available: http://www.westfallteam.com/sites/default/files/papers/risk_management_paper.pdf

[11] O. Mizuno, T. Hamasaki, Y. Takagi, and T. Kikuno, "An empirical evaluation of predicting runaway software projects using bayesian classification," in *Product Focused Software Process Improvement*, F. Bomarius and H. Iida, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 263–273.

[12] E. Wallmüller, "Risk management for it and software projects," 01 2002, pp. 165–178.

[13] M. Felderer, F. Auer, and J. Bergsmann, "Risk management during software development: Results of a survey in software houses from germany, austria and switzerland," 04 2017, pp. 143–155.

[14] T. Hussain, "Risk management in software engineering: What still needs to be done," in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2019, pp. 515–526.

[15] M. Pasha, G. Qaiser, and U. Pasha, "A critical analysis of software risk management techniques in large scale systems," *IEEE Access*, vol. PP, pp. 1–1, 02 2018.

[16] R. Higuera, D. Gluch, A. Dorofee, R. Murphy, J. Walker, and R. Williams, "An introduction to team risk management. (version 1.0)," p. 55, 05 1994.

[17] M. F. Rabbi and K. Mannan, "A review of software risk management for selection of best tools and techniques," 09 2008, pp. 773–778.

[18] M. H. Calp and M. A. Akcayol, "A novel model for risk estimation in software projects using artificial neural network," in *Artificial Intelligence and Applied Mathematics in Engineering Problems*, D. J. Hemanth and U. Kose, Eds. Cham: Springer International Publishing, 2020, pp. 295–319.

[19] Y. Hu, J. Huang, J. Chen, M. Liu, and K. Xie, "Software project risk management modeling with neural network and support vector machine approaches," in *Third International Conference on Natural Computation (ICNC 2007)*, vol. 3, 2007, pp. 358–362.

[20] W.-M. Han, "Discriminating risky software project using neural networks," *Computer Standards & Interfaces*, vol. 40, pp. 15 – 22, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548915000136

[21] C. Kumar and D. K. Yadav, "A probabilistic software risk assessment and estimation model for software projects," *Procedia Computer Science*, vol. 54, pp. 353 – 361, 2015, eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915013654

[22] Y. Hu, X. Zhang, E. Ngai, R. Cai, and M. Liu, "Software project risk analysis using bayesian networks with causality constraints," *Decision Support Systems*, vol. 56, pp. 439 – 449, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923612003338

[23] M. Zavvar, A. Yavari, S. M. Mirhassannia, M. R. Nehi, and A. Yanpi, "Classification of risk in software development projects using support vector machine," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 9, pp. 1–5, 2017.

[24] Y. Hu, X. Zhang, X. Sun, M. Liu, and J. Du, "An intelligent model for software project risk prediction," vol. 1, 12 2009, pp. 629–632.

[25] T. Christiansen, P. Wuttidittachotti, P. Somchai, and S. Vallibhakara, "Prediction of risk factors of software development project by using multiple logistic regression," *ARPN Journal of Engineering and Applied Sciences*, vol. 10, pp. 1324–1331, 01 2015.

[26] Y. Takagi, O. Mizuno, and T. Kikuno, "An empirical approach to characterizing risky software projects based on logistic regression analysis," *Empirical Software Engineering*, vol. 10, pp. 495–515, 10 2005.

[27] C. Molnar, *Interpretable Machine Learning*, 2019, https://christophm.github.io/interpretable-ml-book/.