Defining Technical Risks in Software Development

Vard Antinyan[1], Miroslaw Staron[1], Wilhelm Meding[3], Anders Henriksson[4] Jörgen Hansson[2] and Anna Sandberg[3)]

Computer Science and Engineering [1)] University of Gothenburg | [2)] Chalmers
[3)] Ericsson, Sweden [4)] AB Volvo, Sweden
SE 412 96 Gothenburg

Abstract—Challenges of technical risk assessment is difficult to address, while its success can benefit software organizations appreciably. Classical definition of risk as a "combination of probability and impact of adverse event" appears not working with technical risk assessment. The main reason of this is the nature of adverse event's outcome which is rather continuous than discrete. The objective of this study was to scrutinize different aspects of technical risks and provide a definition, which will support effective risk assessment and management in software development organizations. In this study we defined the risk considering the nature of actual risks, emerged in software development. Afterwards, we summarized the software engineers' view on technical risks as results of three workshops with 15 engineers of four software development companies. The results show that technical risks could be viewed as a combination of uncertainty and magnitude of difference between actual and optimal design of product artifacts and processes. The presented definition is congruent with practitioners view on technical risk. It supports risk assessment in a quantitative manner and enables identification of potential product improvement areas.

Keywords—risk, forecast, measure, software, code, technical, model.

I. Introduction

Managing risks of inefficient product design is of great importance for large software development organizations. The general features of inefficient design are well-known: error-prone or unmaintainable code and models, untestable or untraceable requirements, etc. Chittister and Haimes [1] define the technical risk as the probability and impact of an adverse event. Boehm [2] defines the risk similarly and discuss the top ten risks in software development, most of which are related or directly affect software design. The Software Engineering Institute [3] relies on this definition when outlining the risk management processes. In these studies the definition of risk enables risk quantification by regarding the expected loss as a product of probability and impact of an adverse event. In order to avoid the "probability" element from risk quantification, which is not always possible to estimate explicitly, Barki defines risk as a product of uncertainty and loss [4]. Also, there are others, who view the risk as a qualitative concept that can be estimated subjectively [5, 6].

At present, either classical risk management approaches are applied, which view the risks as a combination of discrete values of probability and impact, or qualitative assessment is performed. However, in practice it is rare to have a discrete value of impact. For example, we cannot assume that a software program either will have an error or will not, because the program might have one, ten or 60 errors, and one error might be acceptable for a product. Similarly we cannot expect a piece of code to be either maintainable or unmaintainable, a requirement to be either traceable or not traceable, etc.

The presence of a continuous component in the risk concept disables quantification of risk exposure as a product of probability and impact of an adverse event. As a result no comprehensive definition of technical risk appears to exist in the field of software engineering, which would permit an effective risk assessment. Therefore an open question remains:

How can we define technical risk in order to support effective risk assessment?

The aim of this paper is to explore the essence of technical risk and define it in a manner that it supports risk assessment and management.

In this study we identified a list of technical risks based on input from four large software development companies; Ericsson, Volvo Car Corporation (CC), Volvo Group Truck Technology (GTT) and Saab. We show that the uncertainty on the difference between actual and optimal designs of product is a key indicator of technical risks; Ones this difference is identified the risk converts into a problem. Provided new definition of technical risk uses the concepts of *uncertainty* and *difference* from optimal design as two continuous measures. Mitigation of such risks becomes a task of identifying and reducing the potentially inefficient design decisions.

The remainder of this paper consists six sections. First we introduce the motivation of this study. Then the definition of technical risk is presented. In section four a list of technical risks, identified in the four companies, are discussed. Section five proposes a risk forecast model for risk assessment. Subsequently we give a brief overview of related studies, and make concluding remarks.

II. Motivation of Study

In our previous research which was initiated by Ericsson, we were requested to create a method for identifying risks associated with source code delivery. That is, to create a method for identifying the most error-prone or hard-to-maintain source code before delivery. The results of research were also applied in Volvo GTT and Saab to identify risky source code [7]. Approximately a year after, we conducted research at Volvo GTT to do similar analysis with textual requirements. The purpose was to identify risky requirements in early development phases in order to permit early reviews and improvements. From the companies' side there is an increasing need for conducting similar analysis on such aspects of development as "identifying risky Simulink models" and "risk based test selection". However, ones the research question was formed, it was unclear how the risk should be regarded. How can we clearly state what is a risk and if the risk occurs what is the impact or the loss? If the risk is viewed as a probability and impact of adverse event it will not be possible to define the impact explicitly, since in practice it does not have a fixed value. For example maintainability, readability and fault-proneness of source code, traceability, feasibility and ambiguity of requirements, efficiency and effectiveness of executed test cases are properties of software artifacts the impact of which are not fixed values. These properties are comparative, which means that a requirement can be more or less traceable, a source code function can be more or less error-prone etc. As there is no explicit value for the impact, the probability of having that impact cannot be defined also. Particularly in our research the risk of low maintainability and high error-proneness could not be viewed as a risk that has distinct probability and fixed impact. This kind of risk could be assessed rather by predicting the possible level of maintainability and error-proneness of the code. One way of doing this could be so: if majority of experienced designers have a strong feeling, that the code consumes twice more maintenance time than they expect, then there is a tangible risk that the organization might spend significant amount of cost over long run of development. Quantitatively assessing different levels of maintainability and error-proneness of the code is not an easy task, but its success can determine the effective applications of proactive decisions.

The presence and increasing role of technical risk assessment in software development, as well as difficulties of adopting classical risk definition for technical risks of software development in practice led us to define the risk in a manner that supports technical risk identification, assessment and mitigation.

III. Defining Technical Risks

In various fields the risk is defined differently. In finance the risk is usually considered as a combination of the probability and the variance of the actual and expected return. In health care, the risk is viewed as a combination of probability and damage. In ISO 31000 the risk is defined as an effect of uncertainty on objectives [8]. All of these definitions contain events which ultimately might have an impact on a person or organization (group of people) that have targeted a specific objective to achieve. In certain definitions people or organizations are not mentioned explicitly, but they are implicitly considered. Generally the risk is defined for a particular person or a group of people and it can influence a particular objective that is targeted to be achieved. In software development organizations technical risks can be described as possibility of undesirable events which ultimately affect software engineers and software development organizations. These risks usually accompany various process or product design decisions and can affect processes and product artifacts.

Prior to defining the technical risk we emphasize three pivotal concepts which will be underlaid in our risk definition:

1. The <u>objective</u> that a person or an organization want to achieve
2. The <u>person</u> or <u>organization</u> that the risk emerges for
3. And their <u>decisions</u> or <u>solutions</u> which can change the impact of risk

The first concept stresses the fact that a certain objective can be influenced by the risk. The second concept indicates that the objective is defined for a person or an organization. If there are more than one interrelated persons or organizations the same risk can affect them differently; defining the risk from one particular person's or organization's perspective creates a possibility of a situation, where reducing the risk exposure for one person might cause an increase of risk exposure for the other person. This might happen because the two persons have different objectives to achieve, and the reduction of the same risk for them might mean making contradictive decisions. The third concept is the taken design decision (solution) which the organization implements in order to develop a particular artifact of the product. It is important to notice that the risk is usually associated with a decision where multiple scenarios are possible. When encountering such situations, it is not always explicit which decision is optimal for a particular person or organization. Moreover, the more uncertainty there is associated with decisions, the more unpredictable the outcomes of those decisions are. In the presence of complete uncertainty there is no way of determining which decision is preferable. While there is always an actual design decision for a particular development operation, we assume that there is always an existing optimal decision for that operation. Furthermore, we consider that for any decision there is a technical risk associated with it; if the taken decision is an optimal solution than the risk exposure is minimal. Making no decision is also a decision that is, leaving the current condition as it is.

Considering that the risk is about possibility of suffering loss in whatever design decision (solution) the organization makes, *the impact of the risk* can be defined as *the magnitude of difference between actual and optimal design solutions*. As long as this difference is not known, the organization takes on a technical risk. The more uncertainty is associated on how much this difference is the more risky the situation is. But ones the organization knows this difference precisely, the risk converts into a problem. By this understanding of risks, for a given person or organization and for a given development operation, when there are several possible design solutions to achieve an objective, we define the risk as:

The technical risk is the degree of uncertainty on the magnitude of difference between the actual and optimal design solutions.

We consider that this *magnitude of difference* is measured either by internal metrics of software quality [9], time or cost of design. The advantage of this definition is that there is a target level of risk minimization which can be achieved by a specific design solution. From this standpoint, addressing technical risk management means to identify the magnitude of difference between optimal and current solutions, and minimize that difference as much as possible. The *magnitude of difference* can be viewed as the *loss* in classic definition of risk. In practice, however, the optimal design solution cannot be determined precisely. Advanced methods and measures might be applied for determining the most preferable design of an artifact, which can be considered as an optimal design.

A symbolic visualization of risk exposure is illustrated in Figure 1. The figure illustrates the concept of technical risk as a combination of the *uncertainty* of the actual non-optimal design solution and the unknown *unnecessary cost* generated as a result of that solution (dark grey area of the figure). When the deficiency of the design solution is identified, the organization has a known design problem (lower-right square). When the current solution is the optimal one but the organization is uncertain about it, it means there is a problem of insufficient assessment, which should show how well the product artifact or process is designed (upper left square). In practice this case is rather rare. If there is an optimal solution and the organization is certain about it by relying on well-established assessment methods, then there is an opportunity of gaining minimal development cost and reusing the current design solutions in the product design.

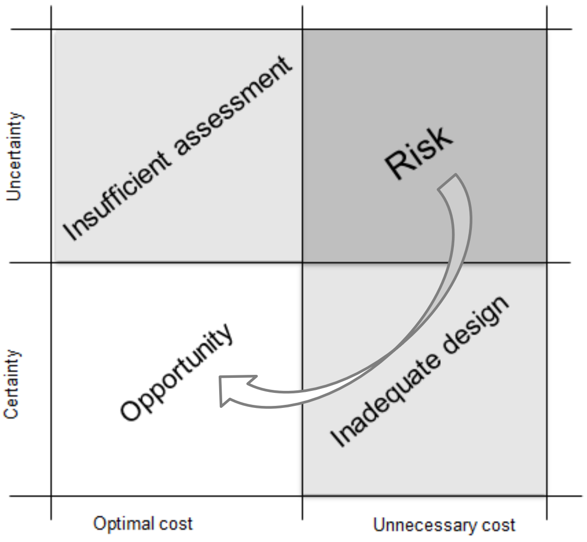

Figure 1 The technical risk as uncertainty on inadequate design

The arch-arrow in the Figure 1 shows the transaction path from *risk* to *opportunity*. Ideally the organization pursues the transaction from risk to opportunity directly but in practice this happens by first identifying the non-optimal design solutions (moving to the low-right area) and then redesigning them (moving to the low-left area). The aforementioned explanation outlines the main characteristic of technical risk: It is the non-optimal design solution of process or product, which the organization is unaware of. In next section we present a list of risks identified at four companies and discuss them.

IV. Software Engineers View on Technical Risks

We discussed technical risks of software development with designers, architects and line managers of large software development organizations in four companies. We organized two company specific workshops on technical risk identification at Volvo GTT and at Ericsson. Subsequently one final workshop was organized with all four companies to identify the main list of technical risks that designers face in software development.

At Ericsson workshops had a specific focus on risks associated with developed source code. At Volvo GTT the purpose of the workshops was identifying main risks specifically associated with requirements implementation and delivery.

The final workshop with all companies was meant to harmonize previously identified technical risks and complement with new ones if there are. The list of all risks, that were identified during the workshops at Ericsson and Volvo and was crystalized during the last workshop with four companies, is presented in Table 1. In the first column of the table we have registered the main description of risks. The risks that are written with bold text are previously identified by other researchers [10, 11]. The risks that are written in italic texts are similar to risks found by other researchers but are not exactly the same.

The descriptions of risks in Table 1 imply that the possible outcomes of these risks are not two discrete values; that is, either the described adverse event will happen or not. All of risks in the table are decisively dependent on product or process design solutions. The better the design solutions are the little the risk is.

Table 1 Main technical risks identified at four companies

N	Risk	Description
1	Multiple “wishes” in one requirement	This requirement contains multiple requests which might make it difficult to develop and test
2	**Inappropriate representation of requirements**	This requirement contains pseudo-code, references to other documents or requirements, etc. which might decrease its understandability from semantic point of view
3	*Untraceable requirements*	This requirement has extensive coupling with other requirements which might make it vulnerable towards outer factors. Late changes of this requirements is highly likely
4	Notes and assumptions in requirements	This requirement contain notes and assumptions which might result in developing wrong functionality
5	**Unfeasible, unclear or untestable requirements**	High likelihood that this requirement is hard to understand and implement because of unclear syntactic description
6	**Requirements’ changes**	This requirement is changed often because of strong business objectives but it might cause risk of late product delivery

7	**Adding requirements in late phases**	This requirement is added after “freezing” the requirements which might delay the product delivery
8	*Inadequate architectural solutions*	This component has unwanted hidden dependencies which might create maintainability and real time performance issues
9	*Simple design mistakes*	For some unclear reasons designers often make simple mistakes in this file. The reasons might be lack of cohesion, non-commented code and complexity. This causes many errors in that file
10	**Unrealistic time and cost estimates**	Because of semantic complexity designers often underestimate the effort in this file. It creates additional efforts for reconsidering the development plans
11	**Gold plating**	This file is a result of an over-flexible design which makes the file difficult-to-maintain
12	**Dependence on external supplied components**	This file is directly and “heavily” dependent on components outside of our control. It might cause late errors
13	**Dependency on other tasks**	This file is directly and "heavily" dependent on execution of tasks outside of its control. It might cause late errors
14	**Real-time performance shortfalls**	This file is vulnerable to real-time performance of the system. There is high likelihood to get post-delivery error reports from the customers
15	*Error-prone, unmaintainable or unmanageable code*	This file is complex and non-cohesive which might increase its error-proneness and decrease the maintainability
16	*Insufficient or ineffective testing*	This test is insufficient; many post-delivery defects might be reported
17	Low number of builds on the integration branch	Low number of builds indicates high likelihood that after integration many defects might appear and impede the smooth delivery
18	Accessing test objects	Missing test objects might create development disruption
19	Queuing	Several processes might be waiting for the completion of this particular process which seems to be delayed.
20	*Underestimating the delivery*	time between “ready” and actual integration of build in main code branch seems to be underestimated which can create high likelihood of late delivery
21	Mismanaging software variants	The likely non-optimal separation or combination of software variants for different platforms creates a risk of additional cost
22	Defect turnaround time’s underestimation	Underestimation of defect turnaround time creates high likelihood for unsatisfied expectations of customers
23	Splitting software into parallel releases	Parallel releases of software might create unexpected additional cost
24	**Missing key-specialist in development**	Certain development process is heavily dependent on a key-specialist the absence of whom can create disruption of development and queuing

For example the source code cannot be either maintainable or not. Instead, the maintainability can be regarded as more or less maintainable. If we quantify the probable loss as consequence of these risk, we get an interval instead of a fixed value, because the more maintainable the code is the less the loss is. Other examples are: (i) increasing changes of requirements causes increasing changes and late errors in a file, (ii) increasing number of “wishes” in one requirement causes decreasing testability and feasibility and (iii) longer queuing time causes higher development cost.

For several risks there is no explicit mitigation strategy that can reduce the risk. These risks contain high level of uncertainty for decision making. For example according to designers, the over-flexible design of the file triggers a risk of undesirable growth of size and complexity; nonetheless, it does not mean that the simplest design minimizes the risk. Practically, at some point the simplification of this file’s design might trigger the increase of risk again, because the error-proneness of the file might increase due to its inability of operating in variety of conditions. Another example is: Volvo GTT produces similar electronic control units for several trucks (Renault, Volvo, Mack, etc.). The question is, should these variants be developed with having completely different requirements design and source code or they can have common design with defined variation points? How this variation points can be determined and how much the development can be carried out jointly or separately? On the one extreme, if everything is separate, the organization might run into big amount of additional costs, because of extensive number of requirements and code that is not reused for all software variants. On the other extreme, there is likelihood that certain part of code and requirements that are different for truck variants, are reused and defects can be left in the software. In this types of risks trade-offs between multiple design choices should be considered.

The risks listed in the table are dependent on design solutions of particular process or product artifact. The design-oriented nature of risks implies that in order to mitigate the risk there is a need of optimal design solutions. In other words the minimal risk exposure can be achieved when optimal solutions are applied. In the next section we discuss the relevance of risk forecast models by application of measurable properties of software product or processes.

V. RISK ASSESSMENT BY FORECAST MODEL

We propose that technical risks can be assessed by risk forecast model which relies on measurable characteristics of software design and processes. The use of software measures as risk predictors are widely suggested [12-14] and software organizations that we collaborated with, where using various measures as technical risk predictors.

Figure 2 shows that the risk has consequences, which can be measured and which the organization would like to minimize. The risk also has indicating characteristics which can be measured for risk assessment. The task is to develop such a mathematical model that using measures of product properties can forecast the consequences of risk. In many cases it is not possible to forecast the consequences precisely as a single number for quality or cost, but an interval of loss can be predicted by certain confidence level. In our previous research [7] for Ericsson we developed a measurement system, which predicted the risk of error-proneness and maintainability of source code.

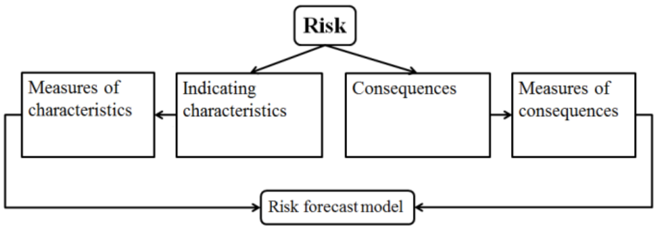

Figure 2 Risk forecast model

The representation of our measurement system by risk forecast model is illustrated in Figure 3. As the figure shows the consequences of risk are additional maintenance time and error-proneness of files. The software engineers at Ericsson had a perception that the maintenance time is higher and the source files are more error-prone than expected. This means that there was initial information about non-optimal design of source code but it was not thoroughly known.

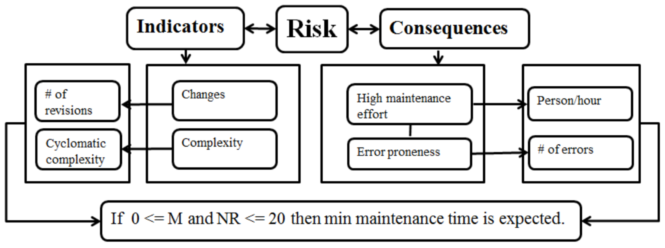

Figure 3 Risk forecast model for Ericsson

In order to assess the risk we identified the risk-indicating characteristics, which were code complexity and change frequency. Complexity and change frequency were measured by calculating cyclomatic complexity (M) and number of revisions (NR) of source code files. Then we designed a model and measurement system based on these measures to assess the risk. Ones the files were assessed, the level of uncertainty on previous design decision of files sharply dropped, revealing the existing problems. Afterwards the designers made decisions of resolving the problem.

In practice, however, there are noticeable problems with assessing what the best design of source code is and how the current design differs from that of optimal design. The identification of optimal design seems to be dependent on more parameters of source code than we are able to measure currently. In foregoing example the superposition (combined metric) of number of revisions and cyclomatic complexity can give an insight how well a source code file is designed, but that combined metric by no means gives a direct estimate on the magnitude of difference between optimal and actual designs of source code file. In fact, what this combined number gives is an estimate how well this particular file is designed compared with other files in the system. We know that from certain point the increasing cyclomatic number and increasing number of revisions indicate decreasing code quality; however this certain point cannot be precisely understood as it can be different for different files. This difference arises from the limitedness of cyclomatic complexity and number of revisions as measures.

VI. Related Work

A comprehensive definition of risk can be found in Kaplan's and Garrick's work [15], where the risk is defined as a superposition of uncertainty and loss. They claim that the risk as a product of probability and consequence is misleading. Furthermore, the authors show that the risk cannot be represented by a single number but rather with a curve, where the risk is not a point on that curve as an expected value but the curve itself.

The term *technical debt* coined by Cunningham [16] is used to discuss the cost that organization pays over time due to non-optimal design of product artifacts. Kruchten, et al. [17] solidifies this term in the context of whole software engineering and define it as “postponed opportunities” or “poorly managed technical risks”. This view is a support for our work, or we may say, our work is harmonious with this view because the difference between actual and optimal design is viewed as the main composite part of the technical risk. Chawan, et al. [18] notice that technical risks emerge because of excessive constraints on the development and poorly defined parameters or dependencies in the organization. Ropponen and Lyytinen [19] conclude that much of the time project risks are not well-understood and their effective assessment and management should be carried out by experienced and well educated managers. We believe that our new definition of risk provides a tangible insight to technical risks so it could be better understood. Bannerman [20] found that the risk management procedures, defined in the literature, do not necessarily support the risk management activities in practice. Bannerman observed that in software engineering on one hand the risk management research lags the needs of practice and on the other hand risk management in practice does not adhere to the research. The current study can be considered as a step for linking general concepts of software risk with its practical needs of management.

VII. Conclusions

In software development organizations continuous identification and assessment of technical risks is an essential activity. The consequences of technical risks can include overall increase of development time and cost, and decrease of pre-delivery product quality. Despite the importance of technical risk assessment, the existing definitions of risks appear not supporting risk assessment. The reason is that in software development technical risks mostly emerge due to inadequate design solutions of processes and product artifacts, but these solutions cannot be regarded as explicit adverse events.

In this research we outlined the essence and main characteristics of technical risks in software development. We showed that while in other fields the risk can be viewed as a product of probability and loss, in software development an explicit occurrence of an adverse event does not exist thus a probability for that occurrence cannot be assessed. Instead we regarded the loss as a continuous variable which is strongly dependent on how good the design solutions of product and processes are. Bigger difference of actual and optimal design solutions and higher uncertainty associated with actual design indicates higher risk exposure. This view of technical risk facilitates the formulation of risk mitigation strategy; (i) identification of optimal design prerequisites and (ii) redesign of product artifacts.

ACKNOWLEDGMENT

We would like to express our thanks to the companies for their support in the study. This research has been carried out in the Software Center, Chalmers, University of Gothenburg, Ericsson, Volvo Group Truck Technology, Volvo Car Corporation and Saab.

REFERENCES

[1] C. Chittister and Y. Y. Haimes, "Risk associated with software development: a holistic framework for assessment and management," *Systems, Man and Cybernetics, IEEE Transactions on,* vol. 23, pp. 710-723, 1993.

[2] B. Boehm, *Software risk management*: Springer, 1989.

[3] R. P. Higuera and Y. Y. Haimes, "Software Risk Management," DTIC Document1996.

[4] H. Barki, S. Rivard, and J. Talbot, "Toward an assessment of software development risk," *Journal of management information systems,* vol. 10, pp. 203-225, 1993.

[5] R. Naik, "Software Risk Management," *International Journal of Advances in Engineering Sciences,* vol. 3, pp. 17-21, 2013.

[6] A. A. M. Chowdhury and S. Arefeen, "Software Risk Management: Importance and Practices," *IJCIT, ISSN,* pp. 2078-5828, 2011.

[7] V. Antinyan, M. Staron, W. Meding, P. Osterstrom, E. Wikstrom, J. Wranker, *et al.*, "Identifying risky areas of software code in Agile/Lean software development: An industrial experience report," in *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, 2014, pp. 154-163.

[8] "Risk Management Principles and Guidelines," *AS/NZS ISO 31000:2009*.

[9] "ISO/IEC 9126, Information Technology - Software product e valuation - Quality characteristics and guidelines for their use, http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749."

[10] T. Arnuphaptrairong, "Top ten lists of software project risks: Evidence from the literature survey," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2011, pp. 1-6.

[11] J. Ropponen and K. Lyytinen, "Can software risk management improve system development: an exploratory study," *European Journal of Information Systems,* vol. 6, pp. 41-50, 1997.

[12] N. E. Fenton and M. Neil, "Software metrics: roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 357-370.

[13] D. E. Neumann, "An enhanced neural network technique for software risk analysis," *Software Engineering, IEEE Transactions on,* vol. 28, pp. 904-912, 2002.

[14] L. E. Hyatt and L. H. Rosenberg, "Software metrics program for risk assessment," *Acta astronautica,* vol. 40, pp. 223-233, 1997.

[15] S. Kaplan and B. J. Garrick, "On the quantitative definition of risk," *Risk analysis,* vol. 1, pp. 11-27, 1981.

[16] W. Cunningham, "The WyCash portfolio management system," in *ACM SIGPLAN OOPS Messenger*, 1992, pp. 29-30.

[17] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software,* vol. 29, 2012.

[18] P. Chawan, J. Patil, and R. Naik, "Software Risk Management," 2013.

[19] J. Ropponen and K. Lyytinen, "Components of software development risk: how to address them? A project manager survey," *Software Engineering, IEEE Transactions on,* vol. 26, pp. 98-112, 2000.

[20] P. L. Bannerman, "Risk and risk management in software projects: A reassessment," *Journal of Systems and Software,* vol. 81, pp. 2118-2133, 2008.