The Requirements Tree Technique for Dependencies-Driven Risk Assessment of AI/ML-based Software Design

Vira Liubchenko[1]

[1]*Odesa Polytechnic National University, 1 Shevchenko Ave., Odesa, 65044, Ukraine*

Abstract

Software requirements engineering is the first phase, which provides the foundation for the software product development process. In software engineering, there are empirical and formal techniques for assessing the requirements specification quality. However, in the case of AI/ML-based systems, the effectiveness of such techniques is low. This problem is related to the high dependence of the system's behavior on data, leading to high requirements uncertainty. Therefore, the risks caused by specific quality requirements for AI/ML components should be dynamically resized concerning design solutions. The paper presents an approach that provides a dependencies-driven assessment of the risks associated with design solutions for AI/ML-based software systems.

Keywords

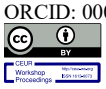
Requirement engineering, risk assessment, AI/ML-based system, requirements tree, software design

1. Introduction

Requirement engineering is the first significant activity in software engineering that involves understanding the problem domain described in a needs statement. This activity performs the transformation of the customer expectation into requirement specifications.

The mining of customer expectations forms the set of requirements that define what the system must ensure in terms of desired effects on the environment for the customer. Instead, the specification describes what software must implement, expressed over the shared phenomena. Usually, there is a gap between the requirements and specifications bridged by assumptions about the behavior and properties of the environment. The assumptions cause risks to the software development process and products; they are critical for achieving requirements. Their ignorance or misunderstanding can lead to a system failure or poor system performance. Therefore, they should be identified and documented as soon as possible.

Analyses of assumptions and respective risks become particularly important in AI/ML-based system development. As it was defined, "AI-based systems are software systems with functionalities enabled by at least one artificial intelligence (AI) component" [1]. ML product is "a system or service for discriminating input data using machine learning technology, such as

CITRisk'2022: 3rd International Workshop on Computational & Information Technologies for Risk-Informed Systems, January 12, 2023, Neubiberg, Germany
EMAIL: lvv@op.edu.ua (V.Liubchenko)
ORCID: 0000-0002-4611-7832 (V.Liubchenko)


neural networks, support vector machines, and linear statistics" [2]. Such systems strongly depend on data, so their behavior is a priory unpredictable and based on assumptions. However, the assumption in AI/ML-based systems could often be violated because

- the unrealistic or missing assumptions due to poorly understood domain and data nature;
- concept drift due to the evolvement of the environment over time;
- adversaries due to deliberately try to violate assumptions;
- feedback loops due to system actions could change the environment over time.

In this paper, based on several publications, we discuss the requirement risk and various risk assessment approaches and techniques in section 2. Section 3 describes the proposed approach based on the requirement tree using. Section 4 presents the validation and results from the analysis of the suggested technique through a case study. Finally, the paper is concluded in section 5.

2. The techniques for software requirements risk assessment

The risks of requirements are the subset of the software project risks. We use the keywords "risk analysis" and "software requirement" to filter relevant works and apply them to the IEEE Xplore collection. Let us describe selected works.

Li and Lui [3] introduced the stakeholder-based approach to risk assessment. They classified all stakeholders into requirement providers and project team members by their responsibilities. As well they distinguished eleven risk factors: six of them – the impact of the system, agreement with requirements, participation level, expression skill, importance extent, and profession level – relate to requirement providers, and five other – technology skill, description skill, stability of work, time pressure, and status – relate to team members. Such detailing makes it easy to find out the cause of the risks. Additionally, the paper introduced quantitative measurement; the risk factors of requirement can be computed as

$$\frac{\sum_1^{n_1} \left(Im \times \sum RRF\right)}{n_1} + \frac{\sum_1^{n_2} \left(St \times \sum PRF\right)}{n_2} \quad (1)$$

Im is a contraction of important extent, and *St* is that of status. *RRF* is a relative risk factor of requirement providers, and *PRF* is that of the project team members. n_1 is the number of requirement providers, and n_2 is that of the project team members. As a result, the numerical assessments for each risk were calculated. Based on numerical estimates, one can prioritize the requirements following the risk but cannot analyze the risk scenarios.

Shaukat et al. [4] provided the means for improving risk prediction at the initial phase of the software development life cycle. They worked with IT experts and mined a wide specter of risk factors grouped into seven attributes: project category, requirement category, risk target category, probability, impact, dimension of risk, and risk priority. The data types of attributes were assigned according to the nature and value set of the data, which provided the possibility to apply the KNN classification technique. The authors demonstrated that the knowledge of the relationship between requirements and risks is necessary for predicting risks in upcoming software projects. The technique requires the existing datasets for training the classifier, which restricts its applicability.

Chandani and Gupta [5] proposed a technique for evaluating risky requirements using the analytic hierarchy process (AHP), which is applied to the requirements' risk from a software project perspective. For evaluation, they used five criteria – impact, perspective, frequency, dependency, and type – with 13 sub-criteria. The authors demonstrated that more decision-making

activities had been catered through the AHP involving risk assessment and prioritization. And as a result, the level of risk for each requirement can be determined.

We should point out that the researchers have regularly returned to using multi-criteria decision-making techniques for requirement analysis. For example, the work [6] describes using AHP for calculating the integral assessment, which encapsulates the risks for every requirement. Another example is the work [7], where requirement risks are evaluated with the TOPSIS method.

All the described approaches encapsulate knowledge about the influence of factors on each other in the estimates. But there are works in which the authors try to keep these dependencies visible.

Gandhi and Lee [8] used their stepwise methodology to discover and understand the correlations among requirements applicable in each operational scenario of the software system to conduct a risk assessment. The methodology is based on Formal Concept Analysis (FCA). The central entity of the methodology is a formal concept defined as a pair of sets (A, B), where A is a set of requirements categories called extent and B is a set of risk components called intent. It is possible to define the order relation on the set of formal concepts. A complete lattice structure called a concept lattice is a partially ordered set of all formal concepts. The concept lattice provides a visual and concise representation of all potential correlations among requirements categories in the given scenario while facilitating their interpretation for risk assessment. The different charts offer the possibility for assessment of different risk aspects.

Towhidnejad et al. [9] presented a study on Software Fault Tree Analysis (SFTA). They pointed out that the main objectives of applying SFTA during the requirement engineering phase are to identify weaknesses in the requirement specification and all the requirements that directly affect the system's safety. Their approach is based on the deductive approach in analyzing events varying from all-purpose to specific circumstances. SFTA has established the potential of differentiating between events based on the AND-OR gate.

Arogundadeet al. [10] also used the tree model, which allowed the study of the scenario's primary paths, and proposed a misuse case-based consequential analysis approach to systematically identify exceptional paths associated with safety hazards using guiding questions. For each scenario path of a use case, authors originally qualified probability and risk. It provides the ground for assigning a probability to a use case by adopting aggregated probability and aggregated risk values.

Muñante et al. [11] defined an engineering framework for risk-driven requirement analysis in self-adaptive systems, which supports risk modeling and estimation during requirements analysis to help designers prioritize risk. Their methodology is based on modeling three aspects: an extended goal model, an environmental model, and a failure model. Let us notice the definition of failure situations as circumstances in which the system cannot explicitly achieve its goals.

If we generalize the considered approaches, all of them assume the deterministic behavior of the system and its components. Also, all techniques evaluate requirements risks separately, focusing mainly on the functional requirement.

In the case of AI-based systems, the behavior is caused by data and cannot be defined precisely as a priory. Respectively the non-functional requirements concerned the property of AI/ML model effect on the successfulness of functional requirements. Therefore, ones need a tool that supports the risk analysis for an "ensemble" of requirements and simulation of risk (or at least risk likelihood) evaluation dynamic propagation.

3. The technique description

In software requirement engineering, one distinguishes two types of requirements. Functional requirements explain how the system must work. In contrast, non-functional requirements describe how the system should perform and are usually defined following ISO/IEC 25010 quality model. The quality characteristics of AI/ML-based systems are an extension of the ISO/IEC 25010 quality characteristics. For example, in [12], the authors described 11 additional features (Table 1).

Table 1

Main quality attributes of AI/ML-based software

Quality attributes	Description
Interpretability	The goal of interpretability is to describe the internals of a system in a way that is understandable to humans.
AI Ethics Requirement	Transparency, accountability, responsibility, and predictability as a subset of transparency are the main principles of AI ethics.
Scalability	The size of input data, the number of variables.
Imperfection	Limited input information.
Safety	Safety in terms of avoiding harm from unwanted outcomes.
Robustness	The effectiveness of algorithms while being tested on new independent datasets.
Complexity	Number of adjustable features (dimensionality of space).
Efficiency	Ability to produce outcomes with a minimum amount of waste.
Fairness	(1) anti-classification, (2) classification parity, and (3) calibration.
Stability	How small changes to its inputs perturb AI/ML.
Staleness	The predictive power of an AI/ML model decreases over time.

It should be noted that the additional quality characteristics are characteristics of the “component” and not the “system” level. The quality model of ISO/IEC 25010 determines which quality characteristics will be considered when evaluating a software product's properties. The quality characteristics of AI/ML determine which quality characteristics will be considered when evaluating a software product's functionality because they affect the quality of results in different scenarios. Thus, the first point for the assessment technique is to consider the dependencies between requirements in risk assessment.

In addition, the AI/ML component introduces an additional complicating factor: a dependence of the component's behavior on data, which makes its behavior non-deterministic. This causes a strong dependence on requirements risks in software design decisions. Therefore, the second requirement for the estimation technique is to consider the degree of the requirement’s support by the chosen design solutions.

The proposed technique consists of two phases: requirement modeling and risk assessment.

Let us describe the sequence of the steps for the first phase.

1. Establish the logical dependencies between requirements. For the analyzed requirement, all functional and non-functional requirements-preconditions are identified. Next, for all identified requirements, precondition requirements are identified. The procedure is repeated until the requirements-preconditions are added.

2. Model the resulting set of requirements and the dependencies between them in the form of a tree.

3. Determine the union operations at the vertices of the bushes of the constructed tree:

3.a. If the bush combines only non-functional requirements, then define the union type as "+" and set the weight coefficients for all arcs emerging from the root of the bush so that the sum is 1.

3.b. Otherwise, define the union type as "*".

This procedure should be performed during the requirements analysis phase. Requirements modeling allows one to understand the requirements content and their importance better.

The risk assessment procedure is performed after the design solutions-candidate are selected. The steps sequence is following:

1. Estimate the level of satisfaction with the requirement when using the selected design solution for every leaf of the requirements tree. We recommend using a scale from 1 to 5, where 1 represents the achievement of some result, and 5 represents the achievement of the best result.

2. Perform a convolution of estimates, moving the tree from bottom to top.

2.a. For the "+" vertex, calculate the weighted sum of the estimates of the vertices connected with it.

2.b. For the "*" vertex, calculate the product of the estimates of the vertices connected with it.

3. The risk factor of the analyzed requirement is calculated as

$$RF = 1 - \frac{Est_{tree}}{Est_{max}}. \quad (1)$$

Est_{tree} is the calculated convolution of estimates, and Est_{max} is the maximum possible estimate.

The value of the risk factor is in the range from 0 to 1, where 0 corresponds to the absence of risk, and 1 corresponds to a situation of complete uncertainty. The decision maker can define thresholds to distinguish between low, moderate, and high risk.

Now let us demonstrate this methodology application by the example.

4. Case study and discussion

Consider a case study of the recommender system (RS), which guides a user in a personalized way to interesting or valuable objects in a large space of possible options or that produces such objects as output [13]. Therefore, the core service of RS is recommendation generation for a particular user.

Any RS needs access to user and item data on which to base recommendations. User data usually are stored in the RS's data storage. Item data can be stored in its data storage or obtained from external sources through APIs. Anyway, the raw data should be processed before being used by an RS. Some RS also need algorithm-specific parameters to perform recommendations well, e.g., a threshold or the depth of a tree. The common data flow in RS is shown in Figure 1.

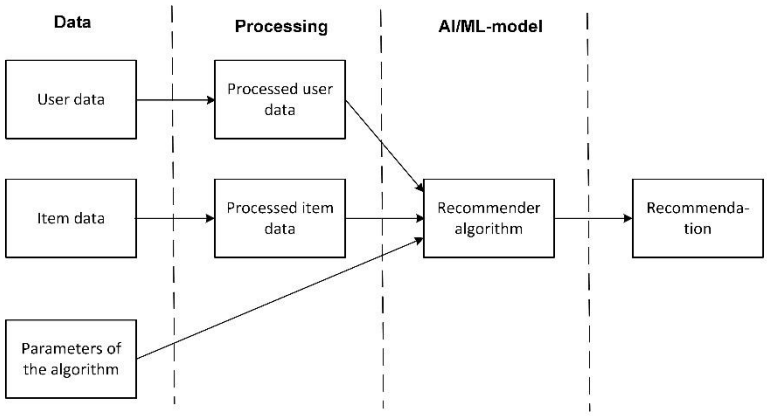


Figure 1: The schema of the recommendation generation process

As we can see, the functional requirement "Provide recommendations" success depends on data processing and respective non-functional requirements (Figure 2):

- Scalability-01: the ability to provide good quality and timely recommendations regardless of the size of the users' dataset
- Scalability-02: the ability to provide good quality and timely recommendations regardless of the size of the items' dataset
- Stability-01: the ability to timely react to changes in user preferences
- Stability-02: the ability to timely react to changes in item availability
- Ethics of data: equality of users
- Robustness: recommendation quality for new users

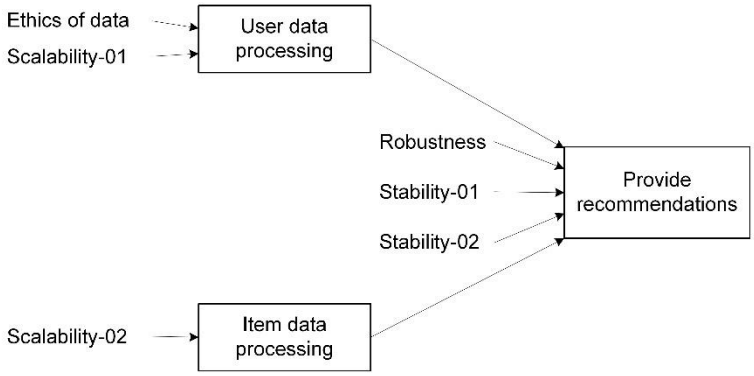


Figure 2: The dependencies between functional and non-functional requirements

The knowledge about dependencies becomes the base for decision tree building. For example, in Figure 3, the requirements tree for the requirement "Provide recommendations" is drawn.

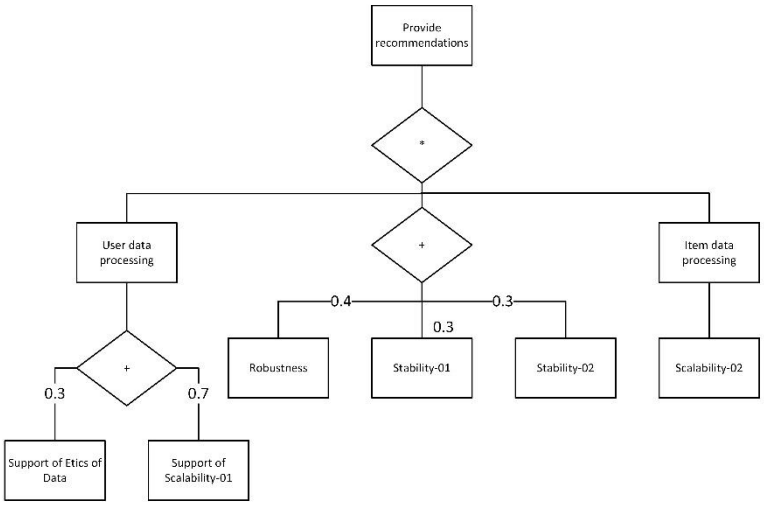


Figure 3: The requirements tree example

While designing the software architecture, there were decided:

- For Scalability-01, all the logic is embedded on the client side, which may take 100 ms to provide recommendations, regardless of the number of users (estimate 5).
- For Scalability-02, all the logic is realized on the server side, which might require 10 ms to recommend one user and 10 secs to recommend 1000 users (estimate 1).
- For Stability-0x, do not provide additional support, which causes the lag in reaction to changes (estimate 3).
- For Ethics of data, do not provide additional support because none of the sensitive features are presented in the datasets (estimate 5).
- For Robustness, realize the feedback loop for the first recommendation for the new user to adapt her descriptors (estimate 4).

This set of design solutions caused a risk factor of 0.872, which responded to the high level of risk. Therefore, the designer should examine her solutions, especially concerning the risk acceptance for non-functional requirements. For example, if there were provided tactics for Scalability-02, which gives the estimate 5, the risk factor would be 0.36, responding to the low level of risk.

As we can see, the technique assesses the risk associated with a particular functional requirement considering the dependencies between the other functional and non-functional requirements. This assessment depends on the design tactics in realizing the requirements set. The decision-making groups (e.g., product managers and software architects) can use the proposed technique for analyzing the different combinations of tactics for program realization.

5. Conclusion

The high dependency on AI/ML-based system behavior from data significantly complicates or makes it impossible to apply most risk assessment techniques. In software engineering, the behavior of a system and its components is assumed to be deterministic. Therefore, we can

evaluate the risks associated with each system requirement and the risks associated with each design solution. Still, design solutions cannot affect the risk assessment.

The additional non-functional requirements specific to AI/ML components have no independent value for the user of the system. However, they have an impact and should be considered in risk assessment system requirements. At the same time, their implementation depends entirely on the available data and the selected algorithms and techniques. Therefore, their risks can only be assessed by considering specific design solutions.

The paper considers an approach that allows assessing the risks associated with functional requirements, considering the dependencies between the requirements and the properties of the adopted design decisions. The technique is helpful in risk analysis of individual functional requirements.

It is known that software design always leads to trade-offs founding both between requirements and between design tactics. Therefore, the following research step covers modifying the proposed technique to assess the risks of several requirements with accounting for the necessary trade-offs.

References

[1] S.Martínez-Fernández, J.Bogner, X.Franch, M.Oriol, J.Siebert, A.Trendowicz, A.M.Vollmer, S.Wagner, Software Engineering for AI/ML-Based Systems: A Survey, ACM Trans. Softw. Eng. Methodol, 31(2), article 37e., 2022. DOI:10.48550/arXiv.2105.01984

[2] Y.Nishi, S.Masuda, H.Ogawa, K.Uetsuki, A Test Architecture for Machine Learning Product, in: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2018, pp. 273–278, DOI:10.1109/ICSTW.2018.00060

[3] X.Li, Q.Liu, Requirement Risk Assessment Focused-On Stakeholder Risk Analysis, in: 2009 33rd Annual IEEE International Computer Software and Applications Conference, 2009, pp.640–641. DOI:10.1109/COMPSAC.2009.199

[4] Z.S.Shaukat, R.Naseemt, M.Zubair, A Dataset for Software Requirements Risk Prediction, in: 2018 IEEE International Conference on Computational Science and Engineering (CSE), 2018, pp. 112–118. DOI:10.1109/CSE.2018.00022

[5] P.Chandani, C.Gupta, Requirement Risk Prioritization Using Analytic Hierarchy Process: A Gateway to Identify Risky Requirements, in: 2018 Eleventh International Conference on Contemporary Computing (IC3), 2018, pp. 1–6. DOI:10.1109/IC3.2018.8530569

[6] V.Liubchenko, N.Komleva, S.Zinovatna, Hierarchical evaluation methodology for software requirements prioritization, in: VIII International Scientific Conference “Information Technology and Implementation" (IT&I-2021), CEUR Workshop Proceedings 3132, 2022, pp. 73–82

[7] C.Gupta, V.Gupta, Requirements Risk Estimation Using TOPSIS Method, in: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), 2020, pp. 505–506. DOI: 10.1109/APSEC51365.2020.00065

[8] R.A.Gandhi, S.-W.Lee, Visual Analytics for Requirements-driven Risk Assessment, in: Second International Workshop on Requirements Engineering Visualization (REV 2007), 2007, pp. 6-7. DOI:10.1109/REV.2007.6

[9] M.Towhidnejad, D.R.Wallace, A.M.Gallo, Fault tree analysis for software design, in: 27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002, pp. 24–29. DOI: 10.1109/SEW.2002.1199446

[10] O.T.Arogundade, S.Misra, O.O.Abayomi-Alli, L.Fernandez-Sanz, Enhancing Misuse Cases With Risk Assessment for Safety Requirements, IEEE Access 8, 2020, pp. 12001–12014. DOI:10.1109/ACCESS.2019.2963673
[11] D.Muñante, A.Perini, F.M.Kifetew, A.Susi, Combining risk and variability modelling for requirements analysis in SAS engineering, in: 2021 IEEE 29th International Requirements Engineering Conference, 2021, pp. 396–401. DOI:10.1109/RE51729.2021.00044
[12] E.Nascimento, A.Nguyen Duc, I.Sundbø, T.Conte, Software engineering for artificial intelligence and machine learning software: A systematic literature review, 2020. DOI: 10.48550/arXiv.2011.03751
[13] A.Felfernig, R.Burke, M.Goeker, Recommender systems: An overview, AI Magazine, 32(3), 2011, pp. 13–18. DOI:10.1609/aimag.v32i3.2361