Innovations in Systems and Software Engineering (2021) 17:17–27
https://doi.org/10.1007/s11334-020-00383-2

ORIGINAL ARTICLE

The realist approach for evaluation of computational intelligence in software engineering

Raghavendra Rao Althar[1] · Debabrata Samanta[2]

Received: 14 September 2020 / Accepted: 19 December 2020 / Published online: 28 January 2021

Abstract
Secured software development must employ a security mindset across software engineering practices. Software security must be considered during the requirements phase so that it is included throughout the development phase. Do the requirements gathering team get the proper input from the technical team? This paper unearths some of the data sources buried within software development phases and describes the potential approaches to understand them. Concepts such as machine learning and deep learning are explored to understand the data sources and explore how these learnings can be provided to the requirements gathering team. This knowledge system will help bring objectivity in the conversations between the requirements gathering team and the customer's business team. A literature review is also done to secure requirements management and identify the possible gaps in providing future research direction to enhance our understanding. Feature engineering in the landscape of software development is explored to understand the data sources. Experts offer their insight on the root cause of the lack of security focus in requirements gathering practices. The core theme is statistical modeling of all the software artifacts that hold information related to the software development life cycle. Strengthening of some traditional methods like threat modeling is also a key area explored. Subjectivity involved in these approaches can be made more objective.

Keywords Software engineering · Data science · Computational intelligence · Software requirements management · Threat modeling

1 Introduction

Exploring software engineering from the data science perspective is the core theme of this paper. Measuring the entropy associated with the software provides a base for studying the patterns in software development processes. Approaches based on metrics, pattern, and anomaly detection systems are adopted for software engineering optimization with data analytics and statistical modeling. Software development processes house various artifacts that are both static and dynamic. Dynamic artifacts like call stacks, application logs, and execution traces are potential sources that can be explored for statistical modeling. Natural Language Processing (NLP) is another area explored in software source code. Other areas explored in the literature include software system's release time performance to ensure software reliability and using stack traces of code crash as a mechanism to identify attack surface identification for the code's security. Based on these code elements' existence on the attack surface and their proximity to each other, can be a good indicator to identify the risk of attacks. These literature findings provide direction for this paper. The integration of all the knowledge hidden in software development processes can be combined with knowledge graphs.

The research area includes the testing process capabilities, such as identifying defects based on historical data and its power to identify defects that lead to the code change, and type of tests based on historical trends. These explorations help build computational intelligence and leverage IR (information retrieval) methods for unstructured data like e-mails. The information in the e-mail can be separated from stack traces, patch-related information, and code. This information is a mix of quantitative data analysis for the software system repositories and qualitative data analysis for the data collated

✉ Debabrata Samanta
debabrata.samanta369@gmail.com

[1] Data Science Department, CHRIST (Deemed To Be University) and Specialist-QMS, American India Private Ltd, First, Bangalore, India

[2] Computer Science Department, CHRIST (Deemed To Be University), Bangalore, India

from the survey. Software debugging can be used as a base for estimation of failure rate, identification of new features, and to assess software reliability. These are some of the prospective areas toward computational intelligence in software engineering. These areas will enhance computational intelligence and contribute to efficient software engineering practices. Security is one of the software industry's top priorities; the same is a core theme for building computation intelligence for software engineering.

2 Problem statement for computational intelligence

The management of security vulnerabilities in software development processes at the requirements gathering phase is the primary problem statement. Identifying influencing factors for software security vulnerabilities, classification of security vulnerabilities, predicting injection of vulnerabilities, and predicting resolution time are other areas that can be explored as an extension of this study. Vulnerabilities identified at the later stage of the software development lifecycle will have a considerable impact on the revenue and customer goodwill. Therefore, the key is to shift the focus and identify the security vulnerabilities as early as possible in the lifecycle. There is a need to understand the software development lifecycle's landscape to figure out what knowledge can be leveraged to tackle these issues. If the problem persists for a long, it will make software systems vulnerable, leading to the loss of customer goodwill and compromised customer data. Research is required to tackle these issues early in the software development lifecycle.

Every time there is a security breach in the computing systems, software companies lose revenue and pay hefty fines in some cases. Security breaches are caused mainly due to vulnerable software. With design loopholes and a lack of an ideal software engineering process, reaching a perfect state is impractical. The problem continues to be part of the system, and an attempt to reduce the impact can be made. Users are affected by a breach of confidentiality and privacy, which makes the software nonusable. This breach affects the industry as they fail to meet the customer needs leading to revenue loss. This problem is generic across geographic locations in software development. If the problem persists, this leads to business loss; users will hesitate to use software applications. The evolution of the technology will find its back seat, as progress will slow down due to the software's security vulnerabilities.

3 Background of computational intelligence

This research has its prominence and recommends using the knowledge already available in the software engineering processes and leverage it to strengthen the processes. When software systems transition from customer requirements to a system deployed in production for the real world, a series of information is created. Information includes issues reported in testing, user-reported issues, and customer change requests. An association of change requests to issues injected, globally acceptable software engineering practices, source code patterns, and other sources also play a crucial role in building computational intelligence. Understanding all these sources of information and collating them will help understand the need for intelligent systems. Promising approaches that have been tried out in the literature provide confidence to leverage some of those data science approaches and expand new techniques to solve these problems.

4 Review of literature

A literature review is conducted on the modeling of software source code for managing software vulnerabilities. In the work [1], the authors identify the significant factors that influence the time taken to fix security issues. Linear regression method, RPART (Recursive PARTioning) method, neural network, and regression methods are explored. Experiments are done on data collected within SAP's secure development process; this data collection process is automatic. One of the key findings of the study is as follows. The fixing time of a security issue is closely associated with the component wherein the potential vulnerability is residing. The project in which the problem is identified, and people involved in issue resolution also influence the time needed for fixing. Interestingly how close the software release date is will impact as well.

Some of the similarities of this research are seen in the work [2]. In this paper, there is a highlight of vulnerability type as one of the influencing factors for the time taken in fixing issues. Work in [3, 4] is similar in finding that the regression technique cannot be superior to other security issues prediction methods. Research in [38] promotes the importance of focusing on injection vulnerabilities being addressed in software and hardware. The study focuses on collaborating software- and hardware-based vulnerabilities injection, providing better insights to device approaches for software vulnerabilities management.

The uniqueness of the research is as follows. Research in the area of effort estimation for addressing security issues is scarce. Research in [5], though focused on the impact of time of fix, but is related to a bug. Work in [1] has security issues as the focus area. Project-related factors that can provide organization factors and people-based factors are covered in this analysis. Information about automating security issues before and after the release is also covered. In comparison with research in [5], a study in research in [1] considers the overall fixed time between a security issue being reported and when the issue is marked as closed. Research in [6] focused on global data covering various projects, whereas this research in [1] is on a specific company and helps get the context of the company's implementation. Study in [7] used static code attributes to learn defect predictors, whereas this research in [1] used the same for assessing the time of issue fix. Research in [16, 17] focuses on end-to-end defect fix time like in [1], but security issue fix time is the focus in research [1]. Researches also use an industrial case study as a base for estimating the time to fix a vulnerability, unlike other research that has focused on open-source software. Data of major software development company are considered, covering security issues before and after the release, covering project and component-based data. Some of the other research in this area includes [9–12]. In the study of [13], the focus is on identifying the influencing factors that impact SAP company's software vulnerability fix time. Research in [14] focuses on two main challenges related to software defect prediction and effort estimation. In this study, an illustration of techniques, tools, and charts is applied to collect, analyze, and model the data. The approach also focuses on the optimization and deployment of the proposed model for various industry settings. This study highlights NB (Naive Bayes) as a better approach than rule-based decision trees. The model used in research [14] is based on research done in [15]. Research in [12] focuses on assessing the value of transfer learning in traditional learning problems that focus on cross-company data and its effectiveness in effort estimation. Result confirms that the transfer learning approach helps to gather knowledge for effort estimation across time. The method used is analogy-based estimation methods for transfer learning. The sample used for the study includes 125 projects developed by a total of 8 companies. The experiment focuses on studying the relevance of decades-old data to ensure that the cost of building those data is justified, and there is no need to spend on building those data systems without return on investment. The study highlights the relevancy filters concept to align the training data to the new application domain and avoid performance loss during transfer learning. They are partitioning data to identify clusters within training data to associate and relate it to test data. Effort estimation of fixing security issues can be explored with transfer learning.

In the research [16], the author reviews management methods for software vulnerabilities with machine learning approaches and data mining techniques. The category of exploration in vulnerability prediction models based on software process metrics, anomaly detection approaches, and recognition of the vulnerable code pattern is covered. Based on a study in research [16] the following observation is made. Vulnerability prediction is being possible with a metrics-based approach, which is a traditional one. Fault prediction models are built for the prediction of vulnerability. Precision improvement for reducing false positives is to focus on future research, while the high recall rates must be maintained as seen in work [1]. As per a study in [18], metrics like code complexity that are development's historical metrics indicate vulnerability discovery. In the study [19], the research is done based on a code review request. Here, the study also highlights the impact of less experienced people's changes to code, which had a crucial role in the injection of vulnerabilities. There is a clear correlation between several code lines that are changed and an increase in vulnerabilities' likelihood. It is also observed that modified files are more vulnerable in comparison with the new files.

Research [16] highlights security issues prediction with software metrics as the base; this approach is yet to mature. Vulnerability-type identification is less explored and is key to helping software engineers prioritize their bug fix based on severity. The granularity of the security issue location beyond at component level, source file, function level, program slice, and execution path level will facilitate more efficient models. Domain-based feature engineering for the software application is explored in the research [20, 21]. Graph mining and graph learning-based feature extraction are studied in [22–24]. Program analysis with deep learning methods though not directly related to software vulnerabilities but can provide an excellent related approach; research in [25] is a useful reference in this area.

Focused feature engineering is an area that needs research; extending to dimensionality reduction will be a suitable combination for further study. Researches are done for dimensionality reduction but with poor results due to poor feature engineering as in the work of [18, 26]. Vectorial and graphical representation studies are the potential areas for exploration. Text mining-based vectorial representation, syntactic structure, control flow information, and data flow information-based visual representation are explored in [21, 27]. Under the metrics-based approach, classification models like logistic regression, Bayesian network research, multilayer perceptron, support vector machines, and random forest approaches have shown similar performance. So, they are not the only primary factor for vulnerability models, dependent on software metrics. Cluster analysis of the data and self-organizing maps are some of the approaches under anomaly detection. These approaches can further experiment with vulnerability management challenges. Comprehensibility of the security-related machine learning model is a prospective area for exploring; some research can be seen in [28–30]. Comprehensibility is about building models that cover all aspects of the area under study and is backed up with enough confidence in the outcome. This research area also contributes to building a comprehensive smart IDE (integrated development environment). Some of the thought processes can be seen in work of [35]. In this research, the authors explore the statistical modeling of software in building intelligent IDEs. Inspiration can be derived from some of the research in the network related to optimizing the network's possible vulnerabilities. Since software development closely interacts with network infrastructure, some of the research approaches can be explored [36].

5 Research gap for computational intelligence

Some of the drawbacks seen in the research [1] are as follows. R-squared metrics are not relevant for nonlinear data but have been used for testing performance measurement only. Though project measures have a strong influence on issue resolution time, there are not enough project-related measures to confirm this. To predict security issues' resolution time, the authors infer no other better regression methods beyond those experimented with within their study. The author also highlights the need for more in-depth research to devise appropriate regression methods that can be the best fit for predicting the time required for fixing issues. It is clarified that the prediction model will not suffice for this modeling as they only provide a static view. The best approach would be to extend the prediction models to account for time evolution, where prediction and forecasting are accounted for together. Limitations around collecting data due to the drastic change in the study period were a key constraint for this study. In research [1], some of the drawbacks noted are evaluation of the concrete influence factors for each stage of the project is planned to be taken in later projects. The study is focused on SAP-specific processes and products. In research [12], drawbacks are the influence of the period selected for data, which is 15 years, may impact the results. The study in [16] highlights the drawback of high false-positive rates in some of the approaches experimented with; this indicates low reliability of these systems. The outputs require a careful manual audit, limiting the usability of anomaly detection systems. Some notable researches have also suffered from high false-positive rates.

6 Research methodology

Approaches explored are under the theme of identifying the software development eco-system patterns that influence introducing issues into the system. Since the system's security is one of the most prominent problems, the focus is on identifying solutions or approaches to discover the pattern of security issues in software development processes. The data science landscape is explored to see what methods fit in well into these problem areas. Also, exploring the features that will help model some of the software system's patterns to enable the software development community with information to make smart decisions and optimize the possible issues that can get injected into the system. These issues are discovered late in the software development lifecycle and hamper the software systems and result in costly loss.

6.1 Understanding software requirements management

Security vulnerabilities in software development are to be discovered as early as possible in the software development process. The requirements phase and design phase will be studied to explore how this early discovery can be made possible. Software requirements phase and design phase-related metrics must be reviewed to understand how this information can assist in security vulnerability management. Identifying the features is a vital part of building effective models for the process. These metrics-based features can be experimented with to create useful models for prediction. There is a focus on understanding various artifacts of software requirements and design phase and the possibility of extracting the features from those. We will explore software source code and other artifacts and derive the pattern from them. Requirements engineering is a significant software development phase; it sets a strong foundation for the final product's quality and effectiveness.

Software metrics help provide insights into the software development process and check how things are progressing and provide feedback to make a required course correction. The project, product, and process-related metrics of software engineering are to be explored. Size, complexity, performance, quality, and features are the composition of product metrics. Project-related metrics govern the ongoing project progress, and process-related metrics look at maintaining and improving software development. Quality metrics are based on the phase in which the product is at any point in time, with end-product and in-process associated metrics. The overall solution for effective and secure requirements management in the software development lifecycle can be envisioned as building the pipeline for handling multiple aspects across the lifecycle, picking the best of the approaches beyond those experimented with earlier to address the weak areas of earlier research. The pipeline will include phases of requirements cleaning or pre-processing, requirements identification, categorization, feature extraction for security requirements, visualization, and data modeling. To add more value to the research outcome, making the machine learning algorithms, a white box in the experiment will be worth it. The usual approach is to use machine learning algorithms as a black box where the focus is on inputs, parameters, and output of the algorithm; in the white box method, the algorithm is explored to the fullest extent to renew it to the specific need of the experiments. This way, it opens the way for efficient utilization of the algorithm and getting the best out of the same. One of the other contributing factors would be to feed the existing knowledge of the software system to strengthen the requirements management's overall pipeline. If the experiments are done on the software systems existing in the production, learning the production systems to feed the requirements management team with critical insights can be handy. Understanding these software systems in production will be possible with statistical modeling of the software source code. Literature provides enough evidence on the source code's naturalness, and source code is more structured than the natural language. These characteristics make it an excellent opportunity to learn from the deployed software systems and feed them back to the requirements management phase for smarter processing. Lessons learned from the life of the software system in production can also add on to this knowledge. These lessons learned may come from reports of production monitoring tools, user feedback, production incidents, and other sources.

6.2 Practitioners perspective on requirements management

There is a basic need to understand the difference between a requirement and an approach in software development. Customer stated requirements are often considered as the final requirements. For example, in banking application security, login authentication is needed, and customer requirements say username to be validated with biometry. Here login authentication is a requirement, and user biometry-based login authentication is an approach. In many cases, requirement managers consider methodology also as a requirement and implement it as stated.

However, it is essential to understand all possible available approaches and assess which is the best one for the scenario. Understanding all possible methods and selecting the best is a good case for statistical modeling to help requirements managers make smart decisions.

In the agile model of software development, requirements are documented as user stories, where requirements specification and acceptance criteria are part of the documentation. Acceptance criteria guide the development team on what is the acceptable scope for the conditions. This aspect is another area where the requirements manager may fail to distinguish between the requirements' approach and acceptance criteria. A significant challenge in requirements management is identifying the implicit and explicit needs. The requirements manager will have the challenge of developing one-line requirements from customers to complete system requirements. Therefore, collaborative research among all the team members of the product team is to develop the requirements covering all the dimensions such as functional, nonfunction including security, implicit and explicit requirements. The requirements management process involves a lot of subjectivity, and it is an art. To make the requirements management phase smart and practical, it needs science to balance subjectivity and objectivity. This balance is where statistical modeling has its role. There are industries developing tools and techniques for requirements management, but there is a need to assess different prospects and concerns about these tools and their adaptability. The agile methodology will influence the requirements management process as the development process is iterative in this model. Requirements gathering, technical solutions, and prioritization are some vital components of requirements management. The business domain also plays a role in understanding the requirements of the company's projects. The domain may not be the deciding factor on the approach applicable to practical requirements management, but this understanding is needed to be validated.

6.3 Statistical modeling for software artifacts

Inspired by the methods employed in research [31], security objectives can be established based on the need for the domain for which software is developed. This aspect is followed by setting the keywords' repository, which will be the language of the software requirements for that domain. It will also include establishing the themes of security for the requirements management phase. In the context of the agile model of software development, requirements are documented as user stories, which are a right-sized piece of the requirement that can be worked upon in a reasonable short period, generally about a week. These user stories will be sized based on the complexity of the requirement, the dependency of the requirements on other requirements, and effort estimates to complete the work. Based on the objectives and critical words derived, these user stories' templates can be established in this context. That will provide a structured way for the requirements manager to navigate the business requirements and translate the conversation with the business into information and security requirements, further transforming security requirements into technical requirements for the application development team to work. This way entire process of requirements management can get streamlined in a customized way specific to the domain. Natural language-based data are involved in requirements management, mostly from the requirements manager's conversation with a customer or business representative. These natural language data can be transformed into the requirements, then classifying the security requirement and then using it to build a framework and models to create an efficient security requirements management process. This part of natural language transformation can take linguistic analysis format; latent Dirichlet allocation (LDA) is one of the topic modeling approaches used in this transformation. In the pipeline of this translation, the word2vec method of Natural Language Processing (NLP) can provide good capabilities for vectorizing the requirements information used for pattern recognition and modeling. Some of the other cutting-edge approaches of NLP can be experiment within this pipeline. Refer Fig. 1 [32]

$$P\left(\theta_{(1:M)}, Z_{(1:M)}, \middle| \beta_{(1:k)} \middle| D; \alpha_{(1:M)}, n_{(1:k)}\right) \tag{1}$$

The posterior joint probability is represented by Eq. (1), which is in play for the latent Dirichlet allocation algorithm. This aspect explains how language modeling is done in some of these Natural Language Processing (NLP) approaches. For M documents with N-words, a single issue generates each of the words out of all topics. θ is the distribution of the topic for every available document. Z represents N number of topics for each document, with β being the distribution of words for each topic. D is the total data available, α is the parameter vector for the document to topic distribution, and n is the parameter vector for each topic to word distribution.

With the statistical modeling of the software source code, it becomes essential to understand the software development eco-systems complex landscape. Since reducing the complexity associated with software development is one of the community's critical objectives, any efforts will add value. On the other hand, approaches used to reduce complexity would provide useful input for software artifacts' statistical modeling. Research in [37] focuses on model-driven engineering to promote models to facilitate the abstraction of information in the software development landscape. Framework for meta-modeling and code generation is an exciting area that can inspire this study.

6.4 Neural network in software security requirements management

The architecture proposed by the work discussed in this article is shown in Fig. 2. Artificial neural network (ANN) can be employed to classify security requirements; some of the NLP approaches can be used at the beginning of the pipeline for processing the input data like text pre-processing as it is a natural language data. One of the other prominent aspects of this study's proposed architecture is the NLP pipeline that manages unstructured customer conversation as its central input. ANN network construction can be done based on the influencing parameter associated with the requirements management phase of software development. Some of the factors that are supposed to be considered for building ANN architecture are the domain of the business, a model of software development, the technology chosen for software development, the skill level of the people involved in requirements management, and the performance of the target application or the software in production. Apart from these, historical learning from issues identified in other phases like design, construction, and testing, and the company or industry's security requirements must be considered. Statistical modeling of these factors will provide the necessary data for this modeling. These parameters are to be feature engineered to make them adaptable to the ANN network. Based on knowledge of the landscape, the ANN network can be customized for specific organizations and projects. If the time-series data is involved, recurrent neural network (RNN) can be a good fit to model the data and feed into the pipeline. Statistical modeling of software source code is another prominent module of this system. This area needs focused research on building a robust system.

Data associated with security will be imbalanced, which means out of the complete set of requirements, actual security-related requirements will be the minority. This aspect

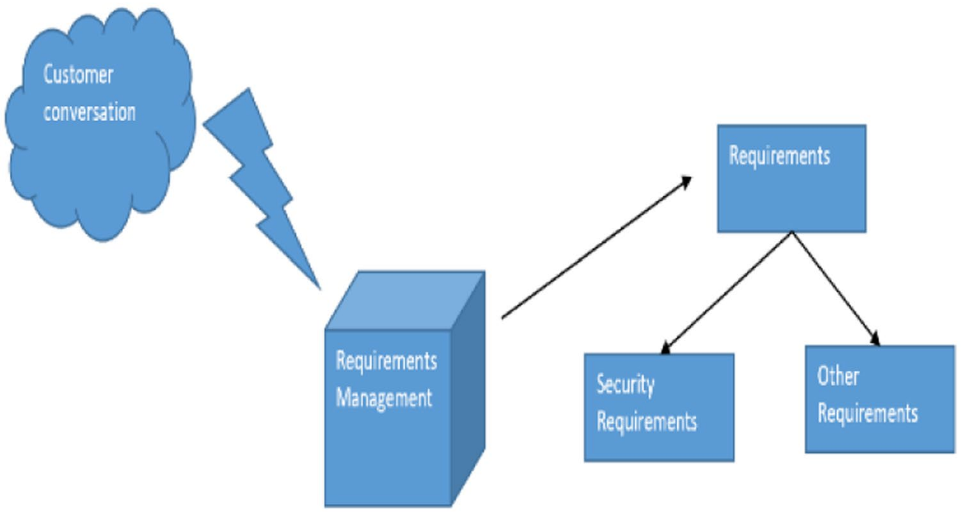

Fig. 1 Requirements management pipeline

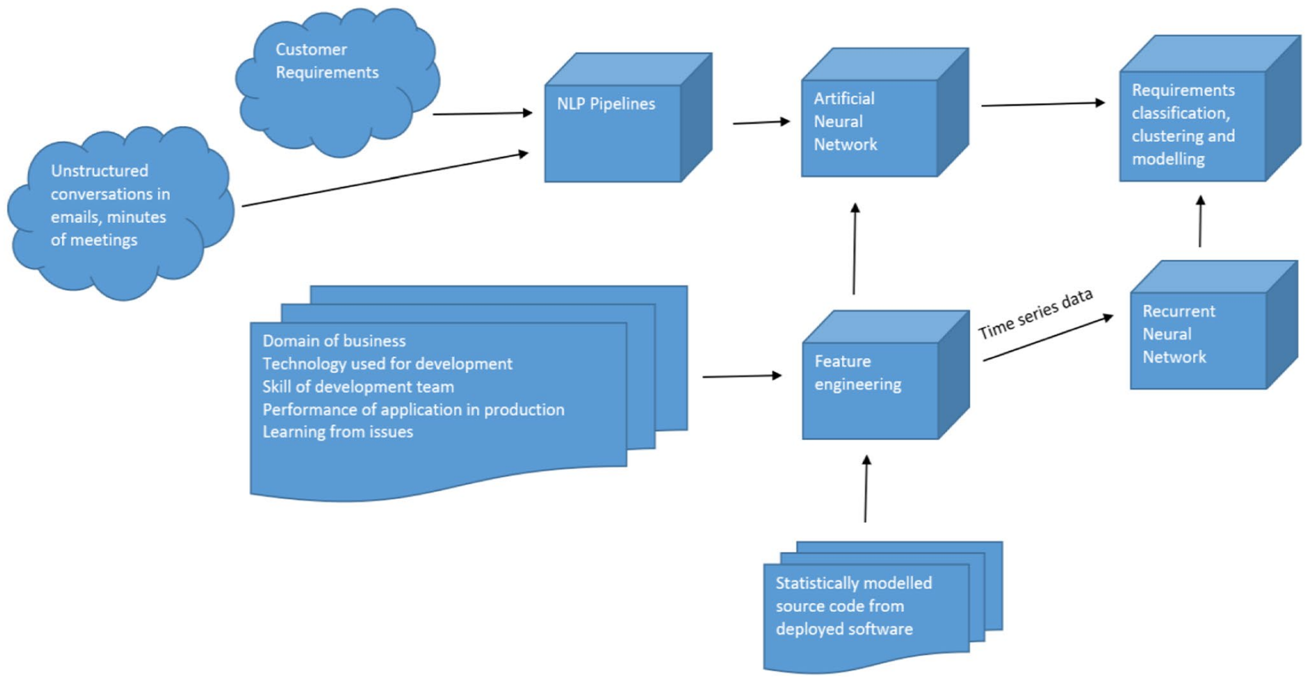
Customer Requirements
Unstructured conversations in emails, minutes of meetings
NLP Pipelines
Artificial Neural Network
Requirements classification, clustering and modelling
Domain of business
Technology used for development
Skill of development team
Performance of application in production
Learning from issues
Feature engineering
Time series data
Recurrent Neural Network
Statistically modelled source code from deployed software

Fig. 2 Proposed requirements management model architecture

points to considering the approaches for data balancing in the experiment. Oversampling, undersampling, and cost-sensitive learning techniques are some of the methods for data balancing. In the case of oversampling, the rare data points are incremented. Synthetic Minority Oversampling Technique (SMOT) is a commonly used method for oversampling. K-nearest neighbor (K-NN) identifies the minority data point in the dataset and synthesizes it for more data. This approach assumes that the data point's relation influences the dataset's local space, but this assumption may not hold good for nonlinearly distributed data. In the case of oversampling, the majority dataset is reduced by using the cluster centric method, which is part of the K-Means algorithm; the focus here is to remove data from majority classes by targeting data points at the end of overlap between various clusters.

6.5 Other approaches for software security requirements management

Much of the research has shown a positive inclination toward the Naive Bayes (NB) algorithm, particularly for text classification involved in the requirements management. There is potential to explore the advanced Bayesian approach for security requirements management. Manual labeling of the data or ranking the data based on expert knowledge would be costly; active learning can be employed to select samples for the experiments on security requirements management. In study in [33], though the focus is on network security, there are closely associated software-related aspects covered as part of this study. The approach of the Bayesian network utilized for exploration of the mutual relationship between vulnerabilities will provide useful insight for extending the thought process toward software security vulnerabilities. Since the landscape of requirements management in the software development life cycle is subjective, more data sources can complement the core requirements management and add value. Other sources of information like memos, e-mails, meeting minutes, and the formal requirements documentation can help. Based on the experts' security issues across the software development lifecycle, the same can be transformed as learned features into the requirements management phase for efficient processing of the requirements. This input helps to understand the security issues pattern across the software development lifecycle and use the knowledge in the software requirement gathering.

6.6 Threats modeling as a source for software security requirements modeling

Threats and vulnerabilities in the system are crucial in understanding how secure the software system is and the focus area for exploration. This threat and vulnerability pair can be an essential feature in learning patterns of security requirements in the software development life cycle. Risk assessment of all the software system components through its evolution across the software development lifecycle will assess the threat and vulnerability pairs. Threat modeling is a well-known approach in the industry. It intends to uncover the prospective loopholes in the software systems and processes involved in their construction. The threat model can be applied across every stage of software development, requirements management, software design, software construction, and software testing, including software deployment, which is a crucial transition activity to realize the real world's requirements. Threat modeling is specialized for the type of software developed, like web-based applications and desktop-based applications. Exploring the missing security controls in the light of the possible threats to the systems and loopholes or the vulnerabilities that can make way for the risks is the intent of threat modeling. Knowing the system is the first step toward threat modeling; this is accomplished with an architecture overview of the system, identifying associated assets involved, data flow diagrams, etc. Since the data flow diagram intends to provide information on how system components interact and work together, exploration of knowledge graphs can optimize the threat modeling if we look at using threat vulnerabilities pair as a critical feature in machine learning models. The study in this paper revolves around making use of all the artifacts in software development processes. The software system's architecture is an essential part of the process; a study in [34], which focuses on pattern identification in software architecture for facilitating the effective choice of architecture, can help provide direction by borrowing some features of the study of secured software requirements management. There is a high potential collaboration between software requirements management and software architecture management to streamline entire software development processes. So, the study in [35] provides a robust feature set. Identification of the threats can be made with some industry approaches like the STRIDE methodology of Microsoft. S stands for Spoofing, T stands for Tampering, R represents Repudiation, I stand for Information disclosure, D means Denial of service, and E stands for Elevation of privilege. This manual exploration process of the threat libraries and picking relevant threats for the operation is a time-consuming and less effective approach. There is the possibility of automating the learning process, where the standard threat libraries are reviewed and learned for the patterns hidden. Also, learning can be done on the target operations to understand the security needs of the systems. The knowledge gathered from the study and recommendations made can be used to ease practitioners' reference for implementation.

6.7 Discussion of results

How do we identify security vulnerabilities in the software requirements management phase? Metrics-based system, threat model-based system, and threat library-based system are the prominent sources of study. Explorations were done on all sources of information that would possibly contribute toward unearthing the issues hidden in the software systems. Metrics that are arrived around the requirements management phase of software development are features that would be used for modeling the security systems of software. The threat model framework was explored to understand its capability to systematically bring out the possible threats that can hamper the system. This approach will also feed in as features for modeling. Industry-wide libraries were also explored to understand the sources from the best practices point of view and the industry's already available learning. This exploration helped bring out a better understanding of the critical features that can be explored in statistical modeling of software artifacts in the light of building computational intelligence for software engineering. Since feature engineering is a core part of statistical modeling, this is an important aspect. Metrics-based statistical intelligence is a state-of-the-art method, but exploring the same from the perspective of building features for statistical modeling is a prospective area, as explored in this research. The threat modeling approach is another state of art explored in software development, primarily in the design phase. This research investigates and proposes the possibility of the same being used in building computational intelligence for the requirements management team. How do we categorize the security vulnerabilities in the software requirements management phase to better understand the issues? Classification and clustering approaches are the traditional machine learning-based approaches. Exploration of this area used feature engineering-based investigation as a base. Then the mechanism of building a framework that will help to structure the requirements management phase is investigated. The pipeline of machine learning and deep learning methods is explored to find a practical solution to build classification and clustering approaches for security vulnerabilities. Some of the basic machine learning approaches are experimented with for the classification of the requirements. The first level of classification is to separate security and nonsecurity-related requirements.

Further security requirements to be subjected to multi-class classification to figure out classes associated with security requirements. Refined approaches for the first level of classification and the second level of classification are further studied. As part of building visibility into the requirement management phase's security requirements, natural language capabilities are being explored to create a text preprocessing pipeline to handle requirements. This pipeline will be refined and customized to project-specific data and domain. Further to this pipeline, insights into requirements are provided using NLP approaches. The idea is to use this module as an initial part of a more extensive knowledge system intended to be built for the requirements management team. Word cloud kind of visualization approaches will be explored as part of this module. RNN (recurrent neural network) is best in class to handle data subjected to change over time. Refinement of the RNN approach with the latest deep learning trend, including transfer learning (TL), will be a prospective exploration area. Some of the preliminary assessments have shown positive signs in this area.

There have been experiments with statistical modeling of the software source code in different contexts, like auto-code suggestions. In this context of building computation intelligence for secure software requirements management, software source code modeling has been explored to feed into the requirements manager's knowledge system. Knowledge graph (KG) is one prospective area being experimented with within this research, both for specific knowledge representation and overall knowledge representation for the requirements management team. Specific details of these experiments will be published in subsequent research.

How do you recommend the best approaches to tackle security vulnerabilities in the software requirements phase, to make sure issues are addressed early in the software lifecycle? The recommender system and knowledge graph (KG) will be the primary exploration area. With the base system of feature engineering and modeling being envisioned, further investigation toward enabling requirements management experts in software development to make smart decisions is the next logical step. All the other possible sources of knowledge from across software development phases are also touched upon, like statistical modeling of software source code. This knowledge helps to understand the software application patterns as it works in the real world. These can provide building blocks toward the recommendations system or a smart requirement management system like the one envisioned by the research team.

6.8 Areas to explore

Area of further exploration includes the study of software requirements vulnerabilities from the view of creating computation intelligence for software development processes. The research hypothesis can consider the following aspects to address some of the historical literature gaps. Prediction models need to be enhanced with the components that will take care of the operations' time dimensions, like the knowledge gathered over time. There should be good synergy in data science models' architecture while handling time-related data and other software development data. When there is an attempt to collect data for a long duration, ways to tackle dynamics associated with the data, such as any significant changes in the processes, must be considered. How do we make sure that the influencing factors selected for the study are all-encompassing and covers all scenarios? Since most of the studies are specific to some companies, how do we establish generalizations for reusing the knowledge across industries? How do we extend the theoretical studies conducted with the publicly available data to industrial settings to assess its practicality? How do we handle rare events in the eco-system and learn from them and use them in our modeling? How do we build an enabling support system to unearth threats and vulnerabilities hidden in the design, which is one of the core parts of this problem? These are the questions that provide a pathway for further research.

6.9 Variables with the sources

Various variables are associated with the security of software requirements management processes. Domain knowledge of the requirements managers, industry standards, and global best practices that highlight the software security issues contribute as variables. The learning is involved in software development across various phases like design, construction, testing, and production. This learning can be in the form of issues identified and improvements made. Software source code associated with the software applications could provide hidden insights about the applications. Proactive initiatives like threat modeling will help to unearth the threat and vulnerability pairs in the system. Global process standards like ISO 27,001 (Information Security Management System) can provide a good data source for the study. We need to consider historical documentation of the finalized requirements, which is a good source of reference. The technology landscape used for application development, the development team's skill, and its performance in production provide variables for modeling and studying the pattern.

6.9.1 Data collection methods

Data collection methods will include exploring publicly available datasets based on some of the datasets analyzed in the literature. Industry-specific data to validate the approach in industrial settings will be needed. Expert interviews to validate some of the data sources and factors included in the statistical modeling of security requirements will hold a key. The focus must be on some of the drawbacks highlighted in the literature, data collection, and interview areas. Though making use of all possible data sources from across the software development landscape is intent, carefully designed studies are needed to validate the relevance of those data points and information, which plays a vital role in the research's success. While we choose the industry sources for learning, their significance for the experiment must be considered.

6.9.2 Analytics tools

We must explore state-of-the-art data science approaches to assess the possibility of answering some of the gaps highlighted in the research gap section. Ensemble architecture of machine learning and deep learning approaches will be needed, as there are multiple sources of data across the software development life cycle. Some data sources support the secure software development mindset in the industry that calls for the ensemble model. A large part of the pipeline at the start involves natural language-based data revolving around the customer and product development team; this opens the need for NLP (Natural Language Processing)-based analytical tools. Since the expert's interview is also a significant part of the research, devising the right conversation tool to collect expert information is essential. This information has to be assimilated to derive insights. As software development's security requirements are assessed, visualization tools will also help build relevant statistical models for building computation intelligence. An optimized threat modeling framework customized for the datasets associated with this study would be a handy tool to enable the proposed system.

7 Conclusion

Secure software development is a potential area to deep dive. There is a need to explore cutting-edge machine learning approaches and deep learning to enable experts to make intelligent decisions and cut some subjective choices. Software engineering is a heavy knowledge domain; there is so much potential to understand patterns in the way processes are conducted; this would provide helpful directions to optimize many other areas in overall software development processes. Since the practitioners, particularly from the product team, are involved in the requirements management, they are surrounded with a large amount of knowledge from industry best practices, compliance requirements, company-specific policies, and fast-growing technological landscapes. There is a need to integrate this knowledge and enable insights. Better leveraging of computational intelligence is needed to make meaningful utilization of all these information sources. This study will also help the product management team to get exposed to other parts of the software development ecosystem. Today, practitioners at requirements management are more aligned with business language, whereas the practitioners across different phases of the software lifecycle are aligned with technical terminologies. This study builds a pathway to establish a collaborative environment between the product team and the software development team. This collaboration creates a stable atmosphere for establishing Technology Business Management (TBM) mindset, where both the product team or business team and technology team converse in techno-business knowledge. Here, the technology team understands business better and proposes a technical solution to a specific business problem. On the flip side, the business team understands the technology and asks for specific technology solutions. That makes a healthy conversation, increases the pace of business and technology growth, and takes care of many loopholes created in the system.

References

1. Othmane LB, Chehrazi G, Bodden E, Tsalovski P, Brucker AD (2016) Time for addressing software security issues: prediction models and impacting factors. Springer nature. Data Sci Eng 2:107–124
2. Othmane L, Chehrazi G, Bodden E, Tsalovski P, Brucker A, Miseldine P (2015) Factors impacting the effort required to x security vulnerabilities. In: Proceedings of information security conference (ISC 2015), Trondheim, Norway, pp 102–119
3. Gray AR, MacDonell SG (1997) Comparison of techniques for developing predictive models of software metrics. Inf Softw Technol 39:425–437
4. Wen J, Li S, Lin Z, Hu Y, Huang C (2012) Systematic literature review of machine learning based software development effort estimation models. Inf Softw Technol 54:41–59
5. Zhang F, Khomh F, Zou Y, Hassan A (2012) An empirical study on factors impacting bug fixing time. In: 19th Researching conference on reverse engineering (WCRE), Kingston, Canada, pp 225–234
6. Menzies T, Butcher A, Marcus A, Zimmermann T, Cok D (2011) Local versus global models for effort estimation and defect prediction. In: Proceedings of the 2011 26th IEEE/ACM international conference on automated software engineering. ASE '11, Washington, DC, pp 343–351
7. Menzies T, Greenwald J, Frank A (2006) Data mining static code attributes to learn defect predictors. IEEE Trans Software Eng 33(1):2–13
8. Hewett R, Kijsanayothin P (2009) On modeling software defect repair time. Empir Softw Eng 14(2):165
9. Ben Othmane L, Chehrazi G, Bodden E, Tsalovski P, Brucker AD, Miseldine P (2015) Factors impacting the effort required to fix security vulnerabilities. In: International conference on information security, Springer Cham, pp 102–119
10. Bener A, Misirli A, Caglayan B, Kocaguneli E, Calikli G (2015) Lessons Learned from software analytics in practice. The art and science of analyzing software data, 1st edn. Elsevier, Waltham, pp 453–489
11. Hamill M, Goseva-Popstojanova K (2014) Software faults fixing effort: analysis and prediction. Technical Report 20150001332, NASA Goddard Space Flight Center, Greenbelt, MD USA
12. Menzies EKT, Mendes E (2015) Transfer learning in effort estimation, empirical software engineering. Empir Softw Eng 20:813–843
13. Javier Lopez, Chris J. Mitchell (Eds.) (2015) Factors impacting the Effort required to fix security vulnerabilities an industrial case study. In: 18th international conference, ISC 2015 Trondheim, Norway, September 9–11, 2015 Proceedings
14. Bener A, Misirli AT, Caglayan B, Kocaguneli E, Calikli G (2015) Lessons learned from software analytics in practice. Elsevier, The art and science of analyzing software data, pp 453–489
15. Misirli AT, Bener AB (2014) Bayesian networks for evidence-based decision-making in software engineering. IEEE Trans Software Eng 40(6):533–554
16. Ghaffarian SM, Shahriari HR (2017) Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a survey. ACM Comput Surv 50(4):1–36

17. Shin Y, Williams L (2013) Can traditional fault prediction models be used for vulnerability prediction? Empir Softw Eng 18(1):25–59
18. Shin Y, Williams L (2011) An initial study on the use of execution complexity metrics as indicators of software vulnerabilities. In: Proceedings of the 7th international research shop on software engineering for secure systems (SESS'11). ACM, pp 1–7
19. Bosu A, Carver JC, Hafiz M, Hilley P, Janni D (2014). Identifying the characteristics of vulnerable code changes: an empirical study. In: Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, pp 257–268
20. Long F, Rinard M (2016) Automatic patch generation by learning correct code. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT symposium on principles of programming languages, pp 298–312
21. Yamaguchi F, Lottmann M, Rieck K (2012) Generalized vulnerability extrapolation using abstract syntax trees. In: Proceedings of the 28th annual computer security applications conference, pp 359–368
22. Charu CA, Wang H (2010) A survey of clustering algorithms for graph data. In: Managing and mining graph data. Springer, pp 275–301
23. Cheng H, Yan X, Han J (2014) Mining graph patterns. Frequent pattern mining. Springer, Cham, pp 307–338
24. Foggia P, Percannella G, Vento M (2014) Graph matching and learning in pattern recognition in the last 10 years. Int J Pattern Recognit Artif Intell 28(01):1450001
25. Peng H, Mou L, Li G, Liu Y, Zhang L, Jin Z (2015) Building program vector representations for deep learning. In: International conference on knowledge science, engineering and management, Springer, Cham, pp 547–553
26. Younis A, Malaiya Y, Anderson C, Ray I. (2016) To fear or not to fear that is the question: code characteristics of a vulnerable function with an existing exploit. In: Proceedings of the sixth ACM conference on data and application security and privacy, pp 97–104
27. Shar LK, Briand LC, Tan HBK (2014) Web application vulnerability prediction using hybrid program analysis and machine learning. IEEE Trans Dependable Secure Comput 12(6):688–707
28. Chorowski J (2012) Learning understandable classier models. Ph.D., Dissertation, University of Louisville. Codenomicon
29. Freitas AA (2014) Comprehensible classification models: a position paper. ACM SIGKDD Explor Newsl 15(1):1–10
30. Van Assche A, Blockeel H (2007) Seeing the forest through the trees: Learning a comprehensible model from an ensemble. European Conference on machine learning. Springer, Berlin, pp 418–429
31. Riaz M, King J, Slankas J, Williams L (2014) Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In: 2014 IEEE 22nd international requirements engineering conference (RE). IEEE, pp 183–192
32. Thushan G (2018) Intuitive guide to latent Dirichlet allocation. https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158. Accessed 10 Dec 2020
33. Deb R, Roy S (2020) Dynamic vulnerability assessments of software-defined networks. Innov Syst Softw Eng 16:45–51. https://doi.org/10.1007/s11334-019-00337-3
34. Kassab M, Mazzara M, Lee J et al (2018) Software architectural patterns in practice: an empirical study. Innov Syst Softw Eng 14:263–271. https://doi.org/10.1007/s11334-018-0319-4
35. Althar RR, Samanta D (2020) Building Intelligent Integrated Development Environment for IoT in the Context of Statistical Modeling for Software Source Code. In: Kumar R, Sharma R, Pattnaik PK (eds) Multimedia technologies in the Internet of Things environment Studies in big data, vol 79. Springer, Singapore. https://doi.org/10.1007/978-981-15-7965-3-7
36. Gomathy V, Padhy N, Samanta D et al (2020) Malicious node detection using heterogeneous cluster based secure routing protocol (HCBS) in wireless adhoc sensor net researches. J Ambient Intell Human Comput. https://doi.org/10.1007/s12652-020-01797-3
37. Idani A, Ledru Y, Vega G (2020) Alliance of model-driven engineering with a proof-based formal approach. Innov Syst Softw Eng 16:289–307. https://doi.org/10.1007/s11334-020-00366-3
38. Given-Wilson T, Jafri N, Legay A (2020) Combined software and hardware fault injection vulnerability detection. Innov Syst Softw Eng 16:101–120. https://doi.org/10.1007/s11334-020-00364-5