A Bird's Eye View on Requirements Engineering and Machine learning

1st Tahira Iqbal
fortiss GmbH
Munich, Germany
iqbal@fortiss.org

2nd Parisa Elahidoost
fortiss GmbH
Munich, Germany
elahidoost@fortiss.org

3rd Levi Lúcio
fortiss GmbH
Munich, Germany
lucio@fortiss.org

***Abstract*—Machine learning (ML) has demonstrated practical impact in a variety of application domains. Software engineering is a fertile domain where ML is helping in automating different tasks. In this paper, our focus is the intersection of software requirement engineering (RE) and ML. To obtain an overview of how ML is helping RE and the research trends in this area, we have surveyed a large number of research articles. We found that the impact of ML can be observed in requirement elicitation, analysis and specification, validation and management. Furthermore, in these categories, we discuss the specific problem solved by ML, the features and ML algorithms used as well as datasets, when available. We outline lessons learned and envision possible future directions for the domain.**

***Index Terms*—Requirements Engineering, Machine learning, State of the art, Overview**

I. Introduction

[1] Machine learning algorithms have been shown to have considerable practical importance in many application domains. This is especially true of domains where large databases are available and a need for exploring some kind of consistency exists or, domains where a program needs to adapt itself to changes [56]. Requirements engineering is a critical part of software engineering and it seems appropriate to use machine learning methods for requirements engineering tasks. Because requirements specification documents are mainly given in natural language, ML can be useful by emulating human processing.

This paper aims to present a survey of how ML benefits existing RE approaches. More precisely, we pursue the following research questions:

RQ1: What is the current state of the practice in ML &RE?

RQ2: What types of learning methods are used when ML is applied to RE?

RQ3: Which are the RE problems that currently use ML methods?

RQ4: Is using ML methods improving RE?

To reply to these research questions, we have performed a literature review, split into data preparation, data collection, and data analysis phase. First, a search string was prepared based on the research questions, then a search was performed over a predefined set of databases and all identified studies were assessed by means of title and abstract. Our literature review is not meant to be an exhaustive study of the field – rather we are offering a snapshot of the current state-of-the-art by borrowing some techniques from Systematic Literature Reviewing.

The major contributions of this article are as follows:

- We provide an overview of the current state of the art of some of the challenges RE faces that may be handled through ML techniques. We focus on two important aspects:
  - Providing an overview of the ML problem categories (classification, regression, clustering, etc.) in use for the support of RE tasks (elicitation, analysis, validation and management).
  - Providing an overview of the common ML models (decision tree, K-Nearest Neighbors, Naive Bayesian, etc.) for tackling RE problems and the data sets if available.
- We analyse the literature to discover trends and lessons on the use of ML in RE.

The paper is organized as follows. The rest of this section provides background information on ML and RE required to understand the remaining of the paper. Section II provides an overview of RE tasks where ML has been used. In section III we summarize the major findings of our study. Finally, in section IV we state the threats to the validity of our work and section V concludes the paper.

A. Machine Learning

Machine-Learning (ML) [35] is a range of algorithms to approximate discover patterns in data. Historically, models and heuristics are human-built exhaustive prescriptions of how a system should behave. ML is grounded on different premises: rather than relying on humans to input all the possible cases the system can handle, the field attempts to extrapolate patterns from a representative set of examples that illustrates expected behaviors. The way in which a learning algorithm operates attempts to emulate the way in which humans learn: from a set of examples, a general model for a behavior is induced.

Many learning algorithms exist, based on different visions of how learning happens in practice [13]. All these algorithms have in common the notion of *features*. Features correspond to characteristics of what is being learned and provide the

[1]This work was supported in part by the European Union's Horizon 2020 Research and Innovation program under grant agreement no. 674875

grounds for the algorithm to abstract from the complexities of the real world. Assume for example that an algorithm should learn, based on a brain scan of a medical patient, to decide whether that patient has brain cancer or not. A number of *features* such as for example the "number of irregular objects in the scan", the "color of such objects", the "disposition of such objects" would be provided to the algorithm. Additionally, the algorithm is provided with a number of brain scans together with annotations that summarize decisions previously taken on them (in our example cancer found / cancer not found). Such datasets are called the *training data*. The learning algorithm then undergoes a *training phase*. It attempts to find an internal model that allows it to map the decisions to the brain scans, given the training data. The model obtained from the training step is useful if it performs well (generalizes) when applied to new data from outside the training set – in our example, when it can accurately diagnose brain cancer for new brain scans. Such generalization is based on the premise that inputs that are "closer", in terms of the given *features*, should lead to "closer" outputs.

The established literature in the domain (e.g. [35]) typically considers three types of machine learning:

1) **Supervised learning:** consists of learning a function using training data including annotations of the outcome of the function to be learned (e.g. patient John Doe with a certain number of physiological characteristics was diagnosed with cancer). Supervised learning can be roughly subdivided in two popular problems: *classification* and *regression*. When the output of the function being learned is composed of categorical values (i.e. classes), then we have a classification problem. The goal is to learn how to link instances or samples for a number of parameters to a certain class of values (e.g. healthy patient or unhealthy patient). However, if the co-domain of the function being learned contains continuous values, then we face a regression problem (e.g. predict the body temperature of a patient given some clinical features of the patient).
2) **Unsupervised learning:** In some cases, the output of the function being learned is not given and we have to find patterns in the training data "blindly." This is called an *unsupervised* learning problem. For instance, one may want to cluster patients based on their symptoms.
3) **Reinforcement Learning:** can be seen as an intermediate problem the co-domain of the function being learned is not given but the procedure is guided nevertheless. In reinforcement learning, an agent has to find a sequence of actions leading to a success. The fact that the sequence leads to a success is not known in advance, but rewards are given to the agent in order for it to know if it follows a path to success.

B. Requirements Engineering

Software systems are developed over millions of lines of code, software modules and documents. The primary goal of a software system is to satisfy its users by proposing functionalities that can meet their needs and expectations. This goal is achieved by applying different methodologies and engineering techniques. One of the key factors to satisfy this goal is to understand and identify the needs of users through software requirements engineering. Software requirements engineering is the process that helps in identifying software requirements in a systematic manner in order to understand what functionalities the targeted system should have in order to fulfill the users' needs.

Software requirements play a key role in the success of a project. In the USA, a survey was conducted over 8380 projects by 350 companies to understand project failure rates. The report's results [47] showed that only 16.2% projects were completed successfully while one-half (52.7%) of the considered projects met with challenges and were only partially completed, with time delays and over budget. Almost 31% of the projects were never completed. The main cause for such failures as identified by executive managers was poor requirements engineering. In detail, main culprits were lack of user involvement (13%), incompleteness of requirements (12%), changing requirements (11%), unrealistic expectations (6%) and unclear objectives(5%).

Software requirements engineering has traditionally four phases; *requirements elicitation*, *requirements analysis*, *requirements documentation* and *requirements verification* [29]. Requirements elicitation [8], [57] helps to understand the stakeholders needs, e.g. what features he/she wants in the software. Requirements elicitation techniques are mostly derived from the social sciences, organizational theory, knowledge engineering and practical experience. For requirements elicitation, different techniques exist in the literature such as interviews, questionnaires or ethnography. The requirements analysis [37] phase emphasizes checking for conflicts and consistency of the requirements. It also makes sure that the requirements are clear and complete. Additionally, the agreed upon requirements are documented in the documentation and verification phase. This documentation has a clear and precise definition of the system functionalities that acts as an agreement between stakeholders and developers. These requirements are documented, usually as natural language, diagrams or mathematically formulae. Such documents are used and iterated upon until the end of the project.

System requirements are classified as business requirements, user requirements, functional requirements (FR) and non-functional requirements (NFR). FR are the system requirements that include the main features and characteristics of the desired system. NFR are the system's properties and constraints [9], [19]. NFR set the criteria for judging the operation of the system e.g. performance, availability or reliability. Business requirements are specified to address business' objectives, vision, and goals. They are defined at a high level of abstraction to preserve the company's proprietary information. User requirements are the users' wish list for the system – they are valuable for ensuring that system performs as the users wished it to.

C. Text Preparation

Requirements are written mostly in natural language. They can appear in a variety of forms such as lists of individual words, sentences, paragraphs, short texts potentially including special characters, or others. Before applying a machine learning algorithm on such data, different steps are employed to transform words into features – such as text mining or natural language processing (NLP). This text preprocessing phase relies majorly on pre-built dictionaries, databases and rules. The common preprocessing steps in the literature we surveyed include tokenization, capitalization, lemmatization, stop words removal, stemming and part of speech (POS). Tokenization is the process of splitting paragraphs into sentences, or sentences into words. Capitalization brings all words in a text to lower case for simplicity. Stop words removal removes connecting words such as "and", "the" or others by comparing the text with a list of stopwords. POS takes text and assigns a part of speech (e.g. a noun, a verb, an adjective, etc) to each word that helps to build the understanding of a text. Stemming is a process where words are reduced to a root by removing the unnecessary suffixes – e.g. "eating" after stemming becomes "eat". Lemmatization is an alternative approach to stemming, which is able to capture canonical forms based on a word's lemma. It uses part of speech and WordNet's lexical database of English for removing inflections. For example, applying stemming to the word "better" fails to provide any lemma, while applying lemmatization to the same word would result in the word "good".

Another way to extract features from a text is the bag of words (BOW) technique. BOW categorizes documents based on a dictionary and occurrences of words. A commonly used BOW method is Vector Space Modeling (VSM). VSM is a way to represent documents in a multidimensional space to allow for information retrieval and classification and clustering of documents. An example of using VSM is to query a corpus in order to find relevant and interconnected (parts of) documents related to specific query terms.

II. CONTRIBUTIONS

A. Requirements Elicitation and Discovery

The manual process of requirement elicitation is expensive in terms of effort and resources. A project's success majorly depends on the precise identification of stakeholder's expectations and requirements for their desired system. A possibility to do requirements elicitation is to mine available datasets, e.g. social media, requirement documents or Apple Store reviews. This mining process is performed with help of different techniques such as NLP and text mining [21] [14]. The latest trend for identifying user requirements is to mine data obtained from platforms such as Twitter, Google Play Store or the Apple Store, by applying ML techniques. While user reviews are not structured requirements, they contain useful information coupled with extra information and noise, which makes manual requirement elicitation a challenging task. Automated requirement elicitation is desirable in these cases to significantly reduce time, effort, and cost. Elicitation from external platforms is mainly a ML *classification* task: given a set of information we wish to identify which parts are it constitute requirements. In the literature we surveyed we notice that also *clustering* is used in auxiliary tasks.

1) Elicitation of Requirements from External Sources: Guzman *et al.* [20] proposed the ALERTme approach for classifying, grouping and ranking tweets during software evolution. Many users share their opinions about various software on Twitter. The large amount of datasets makes it hard to manually identify tweets that contain user requirements. The proposed methodology classifies tweets as improvement requests or not, using the *Naive Bayes* algorithm. To the best of our knowledge, this is the first study of its kind. The classifier was trained according to the following steps: 1) conversion of the pre-processed tweets into a VSM model, 2) training of a classifier on a set of manually annotated tweets, 3) classifying tweets in categories by using the trained classifier. Furthermore, improvement requests were considered during tweet grouping, which helped in sorting and summarizing the requests. The results of the summarization process contained highly ranked tweets based on parameters such as the number of "likes", sentiment, number of shares, among others.

Williams *et al.* [53] operated a similar study on tweets in order to classify them as user requirements. The authors used basic pre-processing techniques and applied the VSM technique on the data. For the learning process, manually annotated (labelled) tweets and the *Naive Bayes* algorithm was used for performing the classification. Based on the results, the authors claim in their work that software tweets are neutral in nature – meaning sentiment analysis did not influence the outcome of the ML algorithm. Also, the work showed improved results when compared to [20]. The study used 4000 randomly selected tweets from ten different software including Microsoft Visual Studio, Google Chrome and Instagram.

Jiang *et al.* [24] mined user reviews from app stores for discovering evolutionary requirements. The authors first extracted opinions about software features from reviews. For automated opinion identification, a syntactic relation-based propagation approach was used for extracting targets and sentiment words iteratively, using known and extracted words. Afterwards, the authors applied k-mean clustering for opinion categorization. The proposed system also helped developers deciding on requirements related to software revenue, by considering economic factors. The work used two datasets of online reviews: one from the Karplersky internet security 2011 software package (from Amazon) with 380 reviews; the other one comprising 461 reviews for the TuneIn Radio Pro V3.6 mobile app (from the app store).

Lange *et al.* [31] used an ML-based software requirement elicitation process while extending an existing military tool called *skiweb*. Different users posted and updated military events and information using this tool. The goal of adding learning capabilities to *skiweb* was mining information from user posts. The proposed recommender system used the supervised *Naive Bayes* algorithm to classify text documents in

order to find related requirements to the post. Furthermore, it used topic modeling to identify the key stakeholders and suggested them other requirements for further analysis according to their interests. This study used an internal organizational dataset Skiweb Data such as wiki and blogs.

Jha *et al.* [23] discovered user requirements by mining app store reviews. The requests were classified into three categories; *bugs*, *features* and *junk*. The methodology proposed by the authors uses the *Naive Bayes* and SVM algorithms. The distinction between types of sentences was identified by frame semantics which, as the name indicates, performs a more semantic classification than the typical syntactic-oriented text classification methods. For each review frames were generated, rather than for words. Due to the small number of features required by frames, a lower dimensional model was produced with enhanced prediction capabilities. The study combined existing datasets from past studies and reviews for iOS apps including CreditKarma, Fitbit and Gmail.

Maalej *et al.* presents in [36] a study on how to classify app reviews as bug reports, feature requests, user experiences or ratings. The authors used a *Naive Bayes* algorithm after comparing several algorithms for classification. They also highlight that binary classifiers performed better than multi classifiers. The work uses a metamodel to enhance the performance of the classification, which includes ratings, tense or sentiment scores. A dataset of 4400 manually annotated reviews from Google Play Store and the Apple App Store were used.

Herrera *et al.* [5] built a recommender system to manage a the participation of a number of stakeholders in the requirements elicitation and prioritization process. In this system, stakeholders can work collaboratively to transform their needs into sets of articulated and prioritized requirements. The system automatically generates specialized topics for building forums for stakeholder collaboration and discussion. The stakeholders' interests are extracted from their user profiles, which also helps in creating recommendations according to the interest of a community of similar stakeholders. In order to identify topics, an unsupervised agglomerative clustering algorithm was applied to unstructured data. The proposed system analyzed online datasets (in natural language) that were gathered from stakeholders. The evaluation dataset was a collection of 36 feature requests created by graduate-level students for an Amazon-like student web-portal system.

B. Requirements Specification and Analysis

Software requirements specifications are usually stated in informal, imprecise and ambiguous natural language, making analyzing them a challenging task. The success of a system does not solely depend on its functional requirements, but also significantly relies on adherence to non-functional requirements. The primary focus of requirements analysis is generally towards the identification and specification of FRs. NFRs are usually identified and specified in later development stages, which can increase the risks of problems during the development lifecycle. NFRs may not be mentioned explicitly in a formal specification requirements documents, even though they exist for all systems in freeform documents such as interview notes or meeting minutes. All types of requirements are analysed differently, and as such, it is useful to separate them. This distinction helps in managing changes in requirements as well as in precisely incorporating them in the development of the system. Manual requirement division into FRs and NFRs is difficult and time-consuming. Machine learning can be useful in supporting analysts in the error-prone task of manually discovering and classifying requirements, having in mind easing further analyses tasks.

1) Identifying Non-Functional Requirements: This is a classification problem, as from a set of requirements we want to decide whether or not requirements are NFRs.

In a study by Slankas *et al.* [46], the authors automatically identify and classify sentences in natural language coming from user agreements, install manuals, regulations, requirements specifications and user manuals into 14 different NFR categories, among which *Access Control*, *Audit*, *Availability* or *Legal*. The authors' two-step process is as follows: 1) parsing natural language; 2) classifying sentences into categories with the k-nearest neighbor algorithm. This led the authors to finding 20 keywords for each category of NFR which are then used as features for their classifier. They subsequently trained the NFR classifier with a wide variety of open and closed source EHRs (Electronic Health Record), various industry standards (HL7, CCHIT) and governmental regulations.

Cleland-Huang *et al.* [7] explored a similar approach and used the k-nearest neighbor classifier for grouping NFRs e.g. *availability*, *look-and-feel* or *legal*. For training their classifier, the authors used 15 requirement specifications developed as term projects by master students at DePaul University.

2) Identifying Functional Requirements: This is a classification problem as from a set of requirements we want to decide whether or not requirements are FRs.

Wang *et al.* [51] applied a combination of ML, NLP and semantic analysis to automatically extract functional requirements and classify them into different types, such as action, objective, goal, temporal or constraints. Their framework employed techniques of semantic role labelling (which assigns a role label for each word in the sentence). The authors trained a variant of Recurrent Neural Network using a E-commerce requirements dataset and tested it on the requirements from the automotive industry. Their approach stemmed from analysing the linguistic characterization of software requirement specifications. The framework consists of 10 FR types and allows capturing semantic information in natural language.

3) Distinguishing Functional from Non functional Requirements: This is a classification problem as from a set of requirements we want to decide whether or not requirements belong to a certain class.

Lu *et al.* [32] automatically classifies text from user reviews for (app) stores into FRs or NFRs. The authors further classify NFRs into four categories: reliability, usability, portability, and performance. The approach used a supervised algorithm called *bagging*. The sentences were augmented by several similar words to the user reviews in the training set. The authors used

6696 raw user reviews from iBooks and 4400 raw user reviews from WhatsApp.

Deoxadez *et al.* [11] use semi-supervised classification techniques for automated classification of FR and NFR from user reviews originally from the app store. This study deals with two problems: 1) minimizing the annotation effort for labelling a big dataset of user reviews, and 2) classification of requirements into FR and NFR. The proposed solution to the first problem used a semi-supervised self-labelling algorithm. Self-labelling algorithms require small datasets to produce results that are comparable to supervised techniques. Features were obtained by applying standard pre-processing and the BOW algorithm. The *Naive Bayes* classification algorithm was used for the second problem. The authors used reviews coming from the top-ranked 40 paid and free apps.

Kurtanovic *et al.* [30] analysed software requirements documents written in natural language in order to classify them as FRs, NFRs and subcategories of NFRs. The authors used the SVM algorithm for the classification. The dataset was imbalanced in terms of FRs and NFRs. For avoiding this problem user comments on software products coming from Amazon were integrated into the main dataset. The study used data from the open source tera PROMISE repository that consists of 625 labeled natural language requirements (255 FR [40.8%] and 370 NFR [59.2%]).

Abad *et al.* [1] targeted two similar problems: the first one is the classification of comments on apps into FRs and NFRs, and the second the classification of NFRs into categories. After pre-processing they increased the weight of influential words in the dataset using feature co-occurrence and regular expressions. The authors then used the decision tree (DT) J.48 algorithm for the classification. Additionally, Binarized Naive Bayes (BNB) was used for sub-classification of NFRs. The study showed that the treatment after the pre-processing approach positively influenced the classification. The approach used reviews from the 40 top-ranked paid and free apps.

Garzoli [17] proposed a system for the analysis of requirements that allowed identifying software functionalities within large collections of requirements written in natural language. It classified the large dataset into five types: FRs, NFRs, *design and construction constraints*, *operator requirements* and *performance requirements*. The goal of the study was to devise a general architecture for large-scale and adaptive requirement analysis. It used BOW together with text mining techniques for inferring lexical and grammatical feature for information retrieval. SVM was used for the classification of requirements. The dataset contained 4,727 annotated requirements, related to three different scenarios and was taken from a naval combat management system.

Wieloch *et al.* presented in [52] Trace by Classification, an ML approach in which a classifier is trained to identify and classify requirements and/or other kinds of software artefacts which occur relatively frequently across different projects. In their research the authors call this process generating trace links for software artefacts in their research. The first step in the approach is to eliminate common stop words. Then, in the training phase the authors identify a set of indicator terms for each NFR category. The classifier is then trained by the set of identified weighted indicator terms which is then used to classify additional artefacts into functional or non-functional requirements (e.g. look-and-feel, performance, security, etc). A probability value represents the certainty that the new requirement belongs to a certain artefact type, computed as a function of the occurrence of indicator terms of that type in the requirement.

4) Requirement Prioritization: Complex software systems generally have thousands of requirements with multiple stakeholders and customers. Each one of them has their own set of requirements and opinions and wants their requirements implementation accordingly. However, factors such as budget or different opinions among stakeholders often make implementing all the requirements a complicated task. Therefore, it is important to make a proper decision for prioritizing requirements considering all the factors that are important for the success of the project. Different models exist in the literature for prioritizing software requirements, among which analytical hierarchical process (AHP) [45], Goal oriented [50] or the cost value approach [27]. In these techniques, human input is very important. Qaddoura *et al.* [40] reviewed prioritization techniques and also shed light on the contribution of ML to the topic. ML can be used for automated analysis of these large set of software requirements prioritization, as well as in helping to improve existing techniques.

Dhingra *et al.* [12] predicted from a portfolio of prioritization methods the most appropriate technique for software requirement prioritization process. The input from the user was taken as characteristic values (detect consistency, maintain information, not available, or both) for different attributes. These attributes list included consistency, traceability, priority basis, rigorous/systematic, distributed stakeholder, cognitive aspects, and human experience. The output was the most appropriate requirement prioritization method e.g. AGORA, AHP etc. The framework proposed has three phases; training phase, fuzzing inference process, and testing phase. The drawback of fuzzy approach was the wrong prediction for boundary values, which was resolved by adopting decision trees. DT learned from datasets and predicted the most suitable prioritization technique. Out of 45 test samples, the framework classified 43 instances accurately.

Avesani *et al.* presented in [3] a study that dealt with the scalability problems that arise in managing the prioritization of a large number of requirements when using the AHP technique. The existing solution to scalability issues used heuristics to decide when the pairwise elicitation process should be stopped. The proposed framework outperformed AHP by giving an accurate approximation of the final ranking while restricting the elicitation effort. It used a rank-based learning algorithm and produced a ranking of all requirements. The input for the learning algorithm were a finite set of requirements, the ranking criteria, initial user preferences and a density function.

A similar study was performed in [4] by the same group [3]

for identifying decision-making issues related to the management of risks in Open Source Software adoption in medium and large organizations. A semi-automated system was proposed that used case-based ranking classification algorithm. The input was priority elicitation of goals by the decision maker and a risk goal ranking function (predefined ranking criteria ordering the goal). As output, it ranked the final risk-based goals.

5) Security Requirements: Due to the orthogonal character of their impact on a system, *security* requirements are notoriously difficult to identify, objectify and quantify [41]. Also during requirement specification, it very often happens that security requirements are masked by FRs (but can be deduced from the context of the domain the system operates in) [44]. Because of this, it often happens in practice that security requirements are only marginally tackled during system construction, paving the way to potentially catastrophic consequences. ML can be of use here by aiding in the identification of segments of text that describe security requirements. This is a *classification* problem: given a text, identify which parts of it correspond to which type of security issues.

Jindal*et al.* [25] automatically learn decision trees that can be used to classify security requirements as *authentication*, *access control*, *encryption* or *data integrity*. Preprocessing of the data is done by stemming relevant terms and the *features* used are such terms.

Riaz and her colleagues [44] use the k-nearest neighbors algorithm to classify sentences in requirements documents as *confidentiality*, *integrity*, *authentication*, *availability*, *accountability* or *privacy* requirements. In order to find adequate sentences and provide context to the classifier, the authors start by finding a type for each sentence among the possibilities *title*, *list start*, *list element* or *normal sentence*. For the classification, the authors use the number of word transformations needed to go from one term in one sentence to a term in another sentence. The classifier is trained using requirement sentences from the healthcare domain that are manually classified. A particularity of the approach is that each security requirement type is associated to a template that helps in translating the security requirements into functional requirements in order to ease during the implementation of the final system.

C. Requirements Validation

Validation is to meant to guarantee that requirements reflect the stakeholders' needs, confirm the quality of the system, its consistency, and traceability.

1) Traceability: In requirements traceability, the emphasis is on the ability to track the lifecycle of requirements and their links with other artefacts. The main barrier to ensure traceability is the effort required for building and maintaining the links between those artefacts. Researchers have thus tried to apply machine learning and automated tools for facilitating the establishment of such links [18]. Traceability is tackled in the research mainly through the use of machine learning classification as well as reinforcement learning methods.

Gervasi *et al.* have investigated in [18] what can be learned from links that are already established. They build classifiers as a mean to develop models of tracing that can then be interpreted by humans in order to understand how requirement tracing is done in practice. Their purpose is to revise existing models of hard-coded traceability tools such as VSM. They used a publicly-available dataset of requirements including traceability information, originally based on the CM-1 project by the NASA Metrics Data Program. Their approach has the following steps: apply preprocessing techniques and transfer requirements into a vector of features, from which a set of classification cases is derived by joining a high-level requirement and one low-level requirement. For every such pair a *link* or *nolink* tag is added, based on whether that particular pair was a true link in the original dataset or not. Finally, they use use the dataset to train and test two different classifiers from the WEKA collection: a Naive Bayesian classifier and the J48 decision-tree classifier.

Sultanov *et al.* [48] find traceability candidates from high-level to low-level requirements by the use of reinforcement learning. They use textual high and low-level requirements documents as input and try to find the candidate traces. Their technique demonstrated statistically significantly better results than the Information Retrieval technique.

D. Requirements Management

1) Visualization: Natural language requirement documents can be hard to comprehend and analyze. Stakeholders have to review and understand requirements for large and complex systems. In these scenarios, basic information visualization techniques such as charts or graphs have been used in requirements engineering. These visualizations are usually applied to textual requirements in order to summarize them. Summarization combines large amounts of information into a single representation for quick consumption by the stakeholders [43]. Machine learning is useful to visualize and to group of large numbers of requirements. In our research, we surveyed both clustering and classification methods were used for this purpose.

The ReCVisu (Requirements Clustering Visualization) tool is presented in [43]. ReCVisu, an exploration tool based on quantitative visualizations helps requirements engineers in understanding the nature of the requirements in a visual form. In ReCVisu, the dependency graph consists of requirements artefacts as nodes and the textual similarities as edges. The automatic grouping of requirements into clusters can help in areas such as uncovering the requirements structure, navigating the requirements space, modularizing crosscutting concerns and understanding requirements interactions and evolution.

Pinqui *et al.* [39] recognize that volume of requirements as big data with which companies struggle to make strategic decisions early on is very large. To aid in solving this problem, they have built a complete visual framework to filter requirements from stakeholders in such a way that architects can better make insightful decisions. They suggest training a multi-class SVM model from domain-specific (mechanics, electronics,

etc.) dictionaries and handbooks. Overall, the authors propose a framework to go from management-oriented to architecture-oriented requirements.

Lucassen *et al.* [33] introduced an automated method for visualizing requirements at different levels of granularity. Their visualization method for user stories consists of the following steps: 1) the generation of an overview which provides a general context for understanding the dataset. The authors have used Word2Vec and Ward's clustering algorithm to build up inter cluster relationship matrix of concepts from the dataset. 2) zooming in and out mechanisms; 3) filtering techniques to reduce data complexity. Possible anticipated applications of this visualization are: discovering missing relationships between clusters that may result in further user stories; teaching system functionality by exploring simplified manageable chunks; and analyzing the expected system changes after new sets of user stories.

2) Structuring Documents: Requirements for a system are usually presented in natural language documents. These documents often require proper structuring for a better overall understanding of the requirements. For this purpose, the document should be organized in independent sections, where each one contains conceptually connected requirements [16]. Moreover, technical reviews are typically used to guarantee a certain level of quality in natural language specifications. However, extensive and comprehensive specifications make it problematic for reviewers to find defects, especially in what regards consistency or completeness. Therefore, ML algorithms can support reviewers in their work by automatically classifying and clustering information that is spread over many sections of many documents [38].

Duan *et al.* [15] used hierarchical clustering for detecting cross-cutting concerns that are beneficial for the process of requirements analysis and architectural design. The authors reported experiments in their work were supported by two tool sets: Poirot, a web-based tool designed to generate traces between various software engineering artefacts which was applied to compute similarity scores between requirements; and a homegrown prototype tool, capable of reading structured requirements specification and generated similarity scores and then clustering requirements.

Winkler *et al.* [54] applied convolutional neural networks to automatically classify the content elements of natural language requirements specifications as "requirement" or "information". The authors claim their approach increases the quality of requirements specifications as it distinguishes content that is relevant for specific software construction activities. For converting natural language into a vector representation the word2vec method is used. A set of 10.000 content elements extracted from 89 requirements specifications of an industry partner were used for training the network through the use of the Tensorflow library, using stochastic gradient descent.

Ferrari *et al.* [16] automatically recognize the sections in a document that should be related or independent in order to enhance the document's structure. The authors have defined a novel algorithm named Sliding Head-Tail Component (S-HTC) for clustering the requirements according to their relatedness (the algorithm is based on known distances – the Jaccard similarity metric, the Levenshtein distance, and the a combination of both). The algorithm groups together similar requirements that should appear contiguously in the document. The effectiveness of the algorithm has been evaluated on a case-study from the railway domain (583 requirements).

According to the work of Rauf *et al.* [42], software specification documents usually contain instances of logical structures, such as business rules, use cases and FRs. Automated identification and extraction of these instances benefits requirements management in fields such as automated traceability, template conformance checking and guided editing. The authors have built a framework that, using requirements documents as an input, attempts to build a template for a general structure of the document. This is achieved by specifying logical structures in terms of their content, textual rendering and variability, and then extracting the instances of such structures from rich-text documents.

Ott *et al.* [38] automatically classified and extracted requirements containing related information, which are spread over many sections of many documents. For such a task, they use the Multinomial Naive Bayes and Support Vector Machines classification algorithms. As input for their studies, they have used two requirements specifications from the automotive domain which describe the functional and non-functional requirements of a DOORS Closure Module. A specification and its referenced documents often sum up to 3,000 pages at Mercedes-Benz (the case-study provider). Their method collects related requirements into classes, which the authors call topic landscape. The authors have built a tool, ReCaRe (Review with Categorized Requirements), an Eclipse-based realization of the topic landscape, including a data connection between IBM Rational and DOORS.

III. DISCUSSION

Our survey work implicitly points to a number of trends that we will concretize and summarize in this section. In general, this study aims to provide a state of the art of the use of ML techniques on various RE tasks for giving an overview to practitioners and act as an entry point to this field for researchers. Note that while the pointers we provide here are informed by the literature review we conducted, this survey is not fully systematic (as described in section IV), meaning our conclusions may be revised and/or extended by future surveys of the domain. Moreover, addressing more details on the specific tasks are out of the scope of this study.

Table I summarizes our findings. It provides partial answers to **RQ2** ("What types of learning methods are used when ML is applied to RE?") and **RQ3** ("Which are the RE problems that are currently using ML methods?") in columns *ML Task* and *themes*, respectively. The table also provides partial responses to **RQ1** ("What is the current state of the practice in ML &RE?") and **RQ4** ("Is using ML methods improving RE?"). The answer to **RQ1** seems to be "at its beginning", given the prevalent lack of comparison with the

state of the art as can be observed in table I. The answer to **RQ4** is "unknown", given that most of the studies read by us were initial proposals with little academic or no industrial validation in real software engineering tools or projects.

Note that table I provides additional information on which types of algorithms are used for each kind of theme, as well as datasets used for learning and which are available online.

It is obvious from our survey that NLP techniques are heavily used throughout a majority of the research tacking the application of ML to RE. This is not surprising and even intuitive. RE is the area of software engineering where natural language is employed more ubiquitously, as RE techniques and tools play the role of interface between stakeholders such as clients, certification entities, architects or developers. Although many attempts have been done to bring formality to requirements engineering (e.g. [34], [49]), the *de facto* language between technical and non-technical stakeholders for real-world projects continues being natural language – in particular, English. The IBM Rational DOORS family [22] of tools is an example of a natural-language based tool for requirement engineering that has become the reference in many domains. In the techniques we have observed, NLP is heavily used for the preprocessing stages of natural language in order to bring the data to a format that can be consumed by a learning algorithm (see section I-C).

The authors of the articles we have processed in our survey point to the idea that ML can potentially bring about enormous benefits in terms of the processing of and taking decisions on large amounts of imprecise and ambiguous data. In the real-world of software engineering, parsing and summarizing requirements is a very time-consuming activity. Also, decisions taken by technical stakeholders are often based on imprecise, incomplete and noisy information and are supported by rules-of-thumb, experience and intuition. ML is by nature built to handle and cope with such challenges – it based on data and it's main purpose is exactly to build model of patterns that humans associate to rules-of-thumb, experience or intuitions. Additionally, ML methods often provide a precise degree of certainty regarding the correctness of decisions taking during a software engineering project. Such measures, although only valid regarding the quality of the learning process, allow assessing the risk associated to certain steps in the course of a project.

A large set of requirements datasets are available online. We have identified a few of such datasets in table I. This fact is a cornerstone for the domain, as most ML algorithms existing nowadays are very data-intensive. One of the authors of this survey has recently written a similar article on the application of ML to formal verification [2], for which the datasets available to learn from are typically very small and almost never made public. The authors of the article recognize that such scarceness of data is partly due to the niche nature of the domain of formal verification, where the datasets are mostly in the form of mathematical proofs. Nonetheless, and in spite of the large body of work regarding the application of ML to formal verification, the rareness of data poses a problem not only to the automated learning, but also to the scientific validation of such proposals. This is not the case in RE, where many datasets are publicly available on which both learning and validation can be done.

The majority of the articles we found on the topic of ML and RE have to do with either the *elicitation* or the *analysis* phases of RE. These findings are compatible with the idea that parsing requirement texts and classifying the information that is contained in them is strenuous for humans and thus it is desirable that such tasks are as automatic as possible. Moreover, we noticed that for requirement elicitation, all the analyzed studies are acquire requirements datasets from external sources (e.g. Twitter, app store, etc) rather internal documents. The *validation* and *management* phases in RE also imply tasks that can be automated as we have shown through our survey, but the state of the art in the domain seems to imply that the first two phases have priority for researchers and practicioners.

Also, we have observed through our readings that while *classification* is the most used ML task, *clustering* also plays an important role in the domain of ML applied to RE. This contradicts the results in [2], where *clustering* has almost no expression in work that applied ML to formal verification. We believe this provides support to the thesis that ML is particularly appropriate to RE, given *clustering* is especially useful when mining non-formal data such as free-form text.

IV. THREATS TO VALIDITY

The validity of our study might be affected by the coverage of the search results, bias on the selection of studies, and inaccuracy of data extraction.

Study Coverage: The study we present here is partial, meaning relevant work could be missing due to inadequate search strings or the list of databases not being complete. The data preparation was based as much as possible on a systematic method, which resulted on a map of the read articles and their main features as relevant to our study.

Study Selection Bias: we understand that the assessment might be biased by the interests of the involved researchers. As such, the themes that we discuss in this article may be influenced by the preferences of the involved researchers. To mitigate for this threat, a set of include and exclude criteria was predefined and researchers assessed the title and abstract of the papers to steer the research. Many of the papers retrieved by our queries apply NLP to requirements engineering but involve no learning (e.g. [55], [10], [6], to cite a few). We have explicitly excluded such papers from our survey: although NLP tools do sometimes include ML algorithms, their functionality is used in a black-box manner by RE researchers and as such they were not taken into consideration.

Inaccuracy of Data Extraction: given the data extraction process might be biased by researcher interest, the selection of data items was strictly driven by the research questions. Moreover, reading assignments were marked by the researchers depending on their confidence level. Low-confidence assign-

	Themes	Contributions	ML Task	ML Model Types	Datasets Used
E	External	[20](○) [53](+) [24](+) [31](○) [23](+) [5](○)	Classification Clustering	(Multinomial) Naïve Bayes Support Vector Machines	Online reviews for KIS 2011 (from Amazon) Skiweb data
S	Non-Functional	[46](○) [7](○)	Classification	k-Nearest Neighbors	Open Source PROMISE Datase[2]
	Functional	[51](○)	Classification	Bi-Directional Long Short-Term Memory Conditional Random Field Network	- -
	Functional & Non-Functional	[32](○) [11](○) [30](○) [1](○) [17](○) [52](○)	Classification	Bagging, Naïve Bayes, SVM	Open Source PROMISE Dataset app-store reviews
	Prioritization	[12](○) [3](+) [4](○)	Classification	Case Based Ranking J.48 DT	-
	Security	[25](○) [44](○)	Classification	Decision-Tree k-Nearest Neighbors	- -
V	Traceability	[18](○) [48](○)	Classification Reinforcement Learning	Naïve Bayes / J48 Decision-Tree	Open Source CM-1 NASA project[3] Open Source Pine Dataset[4]
M	Visualization	[43](○) [39](○) [33](○)	Classification Clustering	Support Vector Machines Ward's method	- -
	Structuring	[15](○) [54](○) [42](○) [16](○) [38](○)	Classification Clustering	Multinomial Naïve Bayes Support Vector Machines Convolutional Neural Networks Sliding Head-Tail Component Clustering Hierarchical Clustering	International Union of Railways (EIRENE Functional Requirements Specification[5]) Mercedes-Benz car development

Legend: (+) improves the state of the art; (-) comparable to or worse than state of the art; (○) no information on how the approach relates to the state of the art

TABLE I: Contributions and ML tasks related to each theme within each RE approach.

ments were discussed between the authors until a consensus was reached.

V. Conclusion

Through our bird's eye view of ML applied to RE we have observed that in the past couple of decades a good amount of research has been done on how to bring these two worlds together. The stakes are high: while requirements engineering is currently a domain under intensive research, attempts to address its challenges academically have translated into few results in practice. Free-form text-based tools with light-weight structuring capabilities such as DOORS are now the norm in practice. Requirements elicitation, analysis, validation and management keep on relying on human expertise and talent. While academics often insist that better formalization brings advantages, the languages in which requirements are formalized do not match the need that stakeholders in the RE process (technical and non-technical) have to communicate through artifacts that are intelligible to all.

While not overstating the potential of ML, which is has its own challenges to overcome such as coarseness of the learned models, overfitting or hungriness for data, we have provided in this article indications that ML might become a cornerstone in RE. For now, it seems like the domain is undergoing a pre-scientific phase: the studies we have analyzed seldom compare themselves with the state-of-the-art (see tableI). This suggests that the current body of research is composed of new ideas, which have not yet been validated to its full extent by the scientific or industrial communities. We thus call for a more extensive survey to validate the preliminary conclusions we present in this work.

References

[1] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider. What works better? A study of classifying requirements. *CoRR*, abs/1707.02358, 2017.

[2] M. Amrani, L. Lúcio, and A. Bibal. ML + FV = Love? A survey on the application of machine learning to formal verification. *CoRR*, abs/1806.03600, 2018.

[3] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Facing scalability issues in requirements prioritization with machine learning techniques. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 297–305, Aug 2005.

[4] P. Avesani, A. Perini, A. Siena, and A. Susi. Goals at risk? machine learning at support of early assessment. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 252–255, Aug 2015.

[5] C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher. A recommender system for requirements elicitation in large-scale software projects. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1419–1426, New York, NY, USA, 2009. ACM.

[6] H. Chen, K. He, P. Liang, and R. Li. Text-based requirements preprocessing using nature language processing techniques. In *2010 International Conference On Computer Design and Applications*, volume 1, pages V1–14–V1–18, June 2010.

[7] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated classification of non-functional requirements. *Requirements Engineering*, 12(2):103–120, Apr 2007.

[8] J. Coughlan and R. D. Macredie. Effective communication in requirements elicitation: A comparison of methodologies. *Requir. Eng.*, 7(2):47–60, June 2002.

[9] A. M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[10] D. K. Deeptimahanti and R. Sanyal. Semi-automatic generation of uml models from natural language requirements. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 165–174, New York, NY, USA, 2011. ACM.

[11] R. Deocadez, R. Harrison, and D. Rodriguez. Automatically classifying requirements from app stores: A preliminary study. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 367–371, Sept 2017.

[12] S. Dhingra, S. G, M. Madan, and M. R. Selection of prioritization technique for software requirement using fuzzy logic and decision tree. In *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–11, Nov 2016.

[13] P. Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015.

[14] L. Dong, X. Zhang, N. Ye, and X. Wan. Research on user requirements elicitation using text association rule. In *Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium on*, pages 357–359. IEEE, 2010.

[15] C. Duan and J. Cleland-Huang. A clustering technique for early detection of dominant and recessive cross-cutting concerns. In *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design*, EARLYASPECTS '07, pages 1–, Washington, DC, USA, 2007. IEEE Computer Society.

[16] A. Ferrari, S. Gnesi, and G. Tolomei. Using clustering to improve the structure of natural language requirements documents. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for*

Software Quality, pages 34–49, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[17] F. Garzoli, D. Croce, M. Nardini, F. Ciambra, and R. Basili. Robust requirements analysis in complex systems through machine learning. In A. Moschitti and B. Plank, editors, *Trustworthy Eternal Systems via Evolving Software, Data and Knowledge*, pages 44–58, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] V. Gervasi and D. Zowghi. Mining requirements links. In D. Berry and X. Franch, editors, *Requirements Engineering: Foundation for Software Quality*, pages 196–201, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[19] M. Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26, Oct 2007.

[20] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20, Sept 2017.

[21] C. Hollis and T. Bhowmik. Automated support to capture verbal just-in-time requirements in agile development: A practitioner view. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 419–422, Sept 2017.

[22] IBM. The IBM Rational DOORS family. https://www.ibm.com/us-en/marketplace/rational-doors.

[23] N. Jha and A. Mahmoud. Mining user requirements from application store reviews using frame semantics. In P. Grünbacher and A. Perini, editors, *Requirements Engineering: Foundation for Software Quality*, pages 273–287, Cham, 2017. Springer International Publishing.

[24] W. Jiang, H. Ruan, L. Zhang, P. Lew, and J. Jiang. For user-driven software evolution: Requirements elicitation derived from mining online reviews. In V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, and H.-Y. Kao, editors, *Advances in Knowledge Discovery and Data Mining*, pages 584–595, Cham, 2014. Springer International Publishing.

[25] R. Jindal, R. Malhotra, and A. Jain. Automated classification of security requirements. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2027–2033, Sept 2016.

[26] H. Kaiya, Y. Shimizu, H. Yasui, K. Kaijiri, and M. Saeki. Enhancing domain knowledge for requirements elicitation with web mining. In *2010 Asia Pacific Software Engineering Conference*, pages 3–12, Nov 2010.

[27] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Softw.*, 14(5):67–74, Sept. 1997.

[28] E. Knauss, S. Houmb, K. Schneider, S. Islam, and J. Jürjens. Supporting requirements engineers in recognising security issues. In D. Berry and X. Franch, editors, *Requirements Engineering: Foundation for Software Quality*, pages 4–18, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[29] G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998.

[30] Z. Kurtanović and W. Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 490–495, Sept 2017.

[31] D. S. Lange. Text classification and machine learning support for requirements analysis using blogs. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, pages 182–195, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[32] M. Lu and P. Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, pages 344–353, New York, NY, USA, 2017. ACM.

[33] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, and M. Saeki, editors, *Conceptual Modeling*, pages 463–478, Cham, 2016. Springer International Publishing.

[34] L. Lúcio, S. Rahman, C.-H. Cheng, and A. Mavin. Just formal enough? automated analysis of ears requirements. In *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, Proceedings*, volume 10227 of *Lecture Notes in Computer Science*, pages 427–434. Springer, 2017.

[35] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[36] W. Maalej and H. Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, pages 116–125, Aug 2015.

[37] B. Nuseibeh and S. Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM.

[38] D. Ott. Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements. In J. Doerr and A. L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, pages 50–64, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[39] R. Pinquié, P. Véron, F. Segonds, and N. Croué. A collaborative requirement mining framework to support oems. In Y. Luo, editor, *Cooperative Design, Visualization, and Engineering*, pages 105–114, Cham, 2015. Springer International Publishing.

[40] R. Qaddoura, A. Abu-Srhan, M. H. Qasem, and A. Hudaib. Requirements prioritization techniques review and analysis. In *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pages 258–263, Oct 2017.

[41] A. Rashid, S. A. A. Naqvi, R. Ramdhany, M. Edwards, R. Chitchyan, and M. A. Babar. Discovering security requirements. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 866–876, May 2016.

[42] R. Rauf, M. Antkiewicz, and K. Czarnecki. Logical structure extraction from software requirements documents. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 101–110, Aug 2011.

[43] S. Reddivari, Z. Chen, and N. Niu. Recvisu: A tool for clustering-based visual exploration of requirements. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 327–328, Sept 2012.

[44] M. Riaz, J. King, J. Slankas, and L. Williams. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 183–192, Aug 2014.

[45] T. L. Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[46] J. Slankas and L. Williams. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16, May 2013.

[47] C. Standish Group. The standish group report, 2014.

[48] H. Sultanov and J. H. Hayes. Application of reinforcement learning to requirements engineering: requirements tracing. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 52–61, July 2013.

[49] S. Teufl. *Seamless Model-based Requirements Engineering: Models, Guidelines, Tools*. PhD thesis, Technical University Munich, Germany, 2017.

[50] A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, RE '01, pages 249–, Washington, DC, USA, 2001. IEEE Computer Society.

[51] Y. Wang and J. Zhang. Experiment on automatic functional requirements analysis with the efrfs semantic cases. In *2016 International Conference on Progress in Informatics and Computing (PIC)*, pages 636–642, Dec 2016.

[52] M. Wieloch, S. Amornborvornwong, and J. Cleland-Huang. Trace-by-classification: A machine learning approach to generate trace links for frequently occurring software artifacts. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 110–114, May 2013.

[53] G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10, Sept 2017.

[54] J. Winkler and A. Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 39–45, Sept 2016.

[55] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 12:1–12:11, New York, NY, USA, 2012. ACM.

[56] D. Zhang and J. J. P. Tsai. Machine learning and software engineering. In *14th IEEE International Conference on Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings.*, pages 22–29, 2002.

[57] D. Zowghi and C. Coulin. *Requirements Elicitation: A Survey of Techniques, Approaches, and Tools*, pages 19–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.