Predicting Requirements Volatility: An Industry Case Study

Anıl Holat[1,2], Ayse Tosun[2]

[1]Aselsan Inc., Ankara, Turkey
[2]Faculty of Computer and Informatics Engineerings, Istanbul Technical University, Turkey

Abstract

Software requirements are exposed to many changes during their software development life-cycle. These changes namely additions, modifications or deletions are defined as requirements volatility. Prior requirement volatility prediction studies utilize different requirement volatility measures. In this study we predict number of changes per software requirement as requirement volatility for a large scale safety-critical avionics project in ASELSAN. We employ a comprehensive metric set to explain requirements volatility: requirement quality measures, project specific factors and requirement interdependencies. Predictive models are created through combining input metric sets with machine learners. Success of models in predicting requirement changes, the best performing input metric combinations, the best performing machine learners and success of models in predicting highly-volatile requirements are evaluated in this study. The best prediction results are obtained with the model employing quality metrics, project specific metrics, network metrics altogether with k-nearest neighbour machine learner (MMRE=0.366). Also the best model correctly identifies 63.2% of highly volatile requirements which are exposed to 80% of the total requirement changes. Our study results are encouraging in terms of creating requirement change prediction tools to prevent requirement volatility risks prior to the requirement review process.

Keywords

Predicting Requirements Volatility, Quality Metrics, Network Metrics, Requirements Quality, Requirements Change

1. Introduction

Although software engineering has experienced significant advancements in the last decades, majority of the large-scale software projects still try to cope with requirement changes during their software development life cycle due to dynamic nature of software development activities [1]. Changes for requirements namely additions, deletions or modifications are defined as requirements volatility [2]. Continual requirement changes during software development have tremendous impact on the cost, the schedule and the quality of the final product. Unfortunately, significant number of software projects cannot be completed successfully or completed partially because of requirements' high volatility [2].

According to a survey conducted by Thakurta [3], project managers use various requirement volatility measures: number of changes to the identified use cases, number of changing requirements identified within the issued change requests, realized requirements out of total requirements, and amount of budget the project had to spent on the changing requirements. Alsalemi et al. [4] also report a literature review on requirements volatility prediction. Accordingly, ten studies have employed machine learning methods to predict requirements volatility until 2017. These studies utilize different requirement volatility measures such as number of requirement changes [5], requirement stability index of the project [6], requirements that will be changed in next iteration [7], requirement change impact [8], the impact of requirements changes on project distribution and cost factor [9], software schedule [10]. Related studies also propose requirements complexity metrics [6], requirement dependency metrics [8], requirement size metrics [5] and requirements evolution metrics [7] to predict their own definition of requirement volatility measure.

In our study we aim to predict number of changes per software requirement by using requirement quality measures, project specific factors and requirement interdependencies. We define requirement volatility as the number of change requests reported for a software requirement. This change request could be either for adding a new requirement or modifying an existing requirement. We chose a safety-critical avionics software project in ASELSAN with more than 20,000 requirements for our study. Loconsole et al. [5] conducted a similar study to predict number of requirement changes using size measures on projects with less than 50 requirements. Our study complements the prior work by mining a larger dataset with thousands of requirements and a more comprehensive metric set considering quality and interdependency aspects of requirements as well as project specific factors. It should be noted that the change requests that we study in this work occured in any phase of software development after Software Requirements Specification (SRS) document has been reviewed and confirmed. Thus we investigate post-SRS requirements volatility for the avionics project under study.

The rest of paper is organized as follows. Section 2

QuASoQ'21: 9th International Workshop on Quantitative Approaches to Software Quality, December 06, 2021, Taipei, Taiwan
aholat@aselsan.com.tr (A. Holat); tosunay@itu.edu.tr (A. Tosun)
0000-0002-8727-6768 (A. Holat); 0000-0003-1859-7872 (A. Tosun)

presents related empirical studies carried on for requirement volatility prediction. Section 3 explains study design model in detail. Results and Threats to Validity of our work discussed in Section 4. Section 5 presents conclusion and points out possible directions for future work.

2. Related Work

In this section we present previous studies that aim to predict volatility for requirements, and we focus on the input metrics they employed. We report details of five relevant studies [11, 6, 7, 8, 5] from the literature review conducted by Alsalemi et al. [4]. We also discuss the approaches of other recently published, related studies [12, 13] in this section.

Nakatani et al. [11] propose a method to predict requirement volatility using social relations between executives, competitors, cooperative organizations, and the natural environment. Those measures can be applied to customer requirements easily but it would take some effort to associate them with software requirements. Christopher et al. [6] present requirements complexity metrics to define volatility. Functional requirement complexity, non-functional requirement complexity, input-output complexity, interface and file complexity measures are used to calculate whole project's stability, whereas we seek to predict requirement volatility for each software requirement. Shi et al. [7] present a model to predict future requirement changes by using previous requirement change metrics. They generated six history metrics for requirements that contain information about volatility of topic, frequency of changes and time duration between changes. History metrics can be used to predict requirements that will be changed in next iteration, but has little use in predicting requirements volatility for new projects. Pedrycz et al. [13] also employ the following change logs as input metrics: created version of requirement, last developer, number of modifications, requirement lifetime duration. Change logs are created on later phases of software development thus they are again not very useful to predict requirements volatility for projects in earlier development phases. Goknil et al. [8] and Hein et al. [12] use requirements interrelations for volatility prediction. Goknil et al. [8] utilize formal semantics of requirement relations as input features, whereas Hein et al. [12] create network metrics by using syntactical natural language data. We have combined both measures and created network metrics by using links between system and software requirements instead of lingual relations between requirement texts. Regarding network metrics we employ degree centrality, eigenvector centrality, closeness centrality and betweenness centrality metrics, whereas Hein et al. [12] used 40 network metrics.

According to the literature review, only one study conducted by Loconsole et al. [5] present an empirical study to predict number of changes per requirement, so this study is the most relevant to our work. Following size measures are used to predict number of requirement changes: number of actors interacting with use cases, number of words in each file, number of revisions of files, number of lines per file. In our study, size measures are also used to represent requirement quality but we also enriched our metric set with project specific metrics and network metrics. It should be noted that we do not take deletion requests and deleted requirements into consideration while defining requirements volatility, because in our industrial context we rarely encounter such requests for the safety-critical software. Finally, we applied our model on more than 20,000 requirements that help us assess the generalizability of our findings on predicting volatility on every software requirement using different metrics sets.

3. Study Design

In this section we explain our empirical study design in detail. In Section 3.1 research questions are explained. The analyzed project for which a model would be proposed is described in Section 3.2. In Section 3.3 selected input metrics for requirement volatility prediction are described. Section 3.4 describes the output measure of the prediction model. The used tools are explained in Section 3.5. Machine learning techniques employed in this study are presented in Section 3.6. Finally in 3.7, performance evaluation measures are defined for our model.

3.1. Research Questions

Our main goal is to predict requirements volatility at earlier stages of development lifecycle, and accordingly two research questions are defined.

Research Question(RQ) 1: To what extent do requirement quality metrics, project specific metrics and network metrics predict the volatility of a software requirement?

Previous studies used different metric sets to predict requirements volatility. In this study we aim to use a comprehensive set of input metrics, and observe their individual effects on requirement volatility prediction. While predicting the volatility, we use the number of change (addition and modification) requests on a software requirement. Being inspired by the metric sets used in the literature, we form a group of requirement quality metrics and network metrics. Additionally, project specific metrics for this particular safety-critical avionics project are defined and utilized throughout this study. During model assessment, the performance of each in-

Table 1
Release based AVPRJ statistics

Release	Number of REQs	Mean CR per REQ	Median CR Per REQ
Release 1	8,640	0.7457	1
Release 2	11,401	1.1165	1
Release 3	2,730	0.5267	0
Total	22,771	0.9051	1

put metric set and the combination of those are reported. Detailed sub-questions related to RQ 1 are also listed below:

RQ 1.1: Which metric group is a better indicator of the number of requirement changes?

RQ 1.2: Which machine learning algorithm is better at predicting the number of requirement changes?

RQ 2: How successful are the proposed models in predicting highly volatile software requirements?

Software requirements have a history of varying number of changes during software development life cycle. Some requirements do not change at all; however, some requirements expose to multiple changes and pose risks to a software project. Practically, our model should predict highly volatile requirements, so those requirements will be reviewed by experienced reviewers in detail. For this research question (RQ 2) we measure the success of our models on highly volatile requirements based on a technique in [14].

3.2. Analyzed Project

We chose a safety-critical avionics software project to perform our analysis. We will refer to this project with AVPRJ in the rest of this paper. AVPRJ has many releases from which three releases are selected. Software requirements for those releases are related since they all belong to the same project; however they are partially distinct since each release consists of implementation of different software components developed by many software developers. AVPRJ has a total of 22,771 software requirements. Some release based descriptive statistic for AVPRJ are given in Table 1. CR is used as an abbreviation for change request, REQ is used as an abbreviation for a single requirement. Most of the employed requirements belong to second release and this release has highest mean change request per software requirement value. Third release has relatively fewer software requirements and less addition or modification is performed on requirements belong to this release. More than half of the requirements are modified at least once for this project; 9,848 out of 22,771 requirements are not changed which complies with Standish Group's survey results over more than 8,000 software projects [15].

3.3. Input Metrics

We have employed several metrics to predict the volatility of each software requirement in AVPRJ. The metrics represent three dimensions: requirement quality metrics, project specific metrics and requirement network metrics. Requirement quality metrics extracted by NASA Automated Requirements Measurement(ARM) tool have been used to predict faulty modules previously [16]. Initially, we believed the way requirements are documented will affect requirements volatility besides fault proneness of modules. Some requirement quality size metrics are already utilized in predicting requirements volatility [5], therefore we decided to include requirement quality metric set in our study. Network metrics are employed to predict requirement change volatility in a recent study [12]. This sparked the idea of utilizing network metrics for requirements volatility prediction. Initial observation of various change request notes confirmed that software requirements that are changed within a particular change request have a tendency to be linked to similar system requirements. Accordingly, we employed network metrics created by traceability information. In order to enrich input metric set with a new metric group we focused on safety-critical avionics project characteristics under this study. Features are evaluated separately and the ones would provide information on requirements volatility are selected as project specific metrics. Rationales of project specific metric selection are given in detail in subsection 3.3.2. Detailed explanations for each group are given in the following subsections.

3.3.1. Quality Metrics

While selecting requirement quality metrics to predict volatility, we have inspired by two studies. The first study propose requirement metrics in the context of NASA Metrics Data Program(MDP) to predict software faults [16]. These metrics are calculated by automatically going through requirements documents to highlight vague, ambiguous, long, complex requirements. The second study also reports requirement quality metrics [17] to find out which requirement quality analyze tool is more successful regarding measurement of those metrics.

Combining both studies' list and customizing that to the requirements document templates in our industrial context, we present 20 quality metrics in Table 2. All of these metrics take numeric values, e.g. number of flow sentences in a requirement, number of directives in a requirement.

During the preprocessing, stage, we had to remove three metrics from our analysis since they gave little to no information for AVPRJ: Conditional, Rationale and Subjective. Only one requirement contains conditional expressions, three requirements contain rationale expres-

Table 2
Requirement Quality Metrics

Acronyms	The number of abbreviations in a software requirement. For AVPRJ permitted acronym list is used to extract this metric.
Actions	The number of actions to be performed if conditions of a software requirement are satisfied.
Ambiguity	The number of ambiguous expressions in a software requirement, e.g. adequate, sufficiently, optimal, slow.
Chars between punctuation	Average character count between punctuation marks. Long sentences without punctuation marks decrease readability.
Conditions	The number of conditions need to be satisfied to perform a software requirement.
Conditional	The number of phrases that give the developers freedom to whether or not to implement a software requirement, e.g. maybe, can't, would.
Connectors	The number of connectors that are employed to link multiple sentences or group of words, e.g. and, or, as well as.
Directives	The number of directive expressions to refer a table, a note, a figure or an example.
Flow sentences	The number of expressions that semantically bond a sentence to another one, e.g. although, but, else.
Imperatives	The number of phrases that command to perform particular actions in a software requirement, e.g. shall, must, will.
Implicitness	The number of pronouns that make the software requirement difficult to understand, e.g. this, that, it. A software requirement should be defined explicitly.
Incompleteness	The number of expressions that indicate a software requirement is yet incomplete, e.g. and so on, tbd, etc.
In links	The number of incoming links to a software requirement from other documents. For AVPRJ test cases are linked to software requirements, so the number of in links refer to the number of linked test cases.
Negative Sentences	The number of phrases that give negative meaning, e.g. doesn't, none, can't.
Nested levels	For AVPRJ nested level metric value is the greatest level in hierarchical nesting structure of a software requirement.
Out links	The number of out links of a software requirement. In AVPRJ software requirements are linked to system requirements. Therefore the number of out links is the total number of linked system requirements by a software requirement.
Rationale	The number of expressions that give justification in a software requirement, e.g. thus, in order to.
Speculative Sentences	The number of speculative phrases which lead to question necessity of a software requirement, e.g. normally, eventually, almost.
Subjectivity	The number of subjective expressions presenting personal opinion rather than objectivity e.g. I think, in my opinion.
Text length	The total number of characters in a software requirement.

sions and none of the requirements have subjective expressions. Thus we ended up having 17 metrics representing the quality aspect of requirements for predicting their volatility.

3.3.2. Project Specific Metrics

Project specific metrics may differ regarding the scope of a software project, but the metrics we chose to use are not so specific to the development environment, programming language, or domain in which the software is developed. We believe project specific metrics would provide information about development characteristics in an organization, and hence the factors affecting the change proneness of requirements. Table 3 list these project specific metrics employed in this study. If the project follows an inspection activity on requirements, it is more likely that the team would find the ambiguities and inconsistencies on the requirements. Since derived requirements are not part of customer needs, they cannot be validated through user acceptance tests. If a requirement has a safety aspect, more comprehensive software tests will be performed, thus exposure of a potential change is highly probable. Number of related components is a measure of impact of a software requirement on general product, thus more feedback will be given to requirements affecting many components by development team. Each software release has different dynamics that affect requirements maturity e.g. release schedule, experience of developers, complexity of system. For example if sched-

Table 3
Project Specific Metrics

Inspection	Indicates if a software requirement is evaluated through an inspection activity. This procedure might be preferred to complement functional tests.
Derived	Software requirements that are not explicitly stated in system requirements but derived based on design decisions [18].
Safety	Shows if a software requirement is safety critical.
No. of Related Components	Number of isolated software components that a requirement is related.
Release Number	Release number that the software requirement belongs to.

Table 4
Network Metrics

Degree centrality	Gives score to requirements based on the number of links.
Betweenness centrality	Measures how many times a requirement is on the shortest path in the graph.
Closeness centrality	Indicates how close a requirement to other requirements considering the whole graph.
Eigenvector centrality	Measures how a node influences other nodes in network through connections.

ule is too tight to complete SRS document, requirements could be immature and more requirements changes could be performed in the future for this release.

3.3.3. Network Metrics

Hein et al. [12] earlier utilized 40 network metrics to predict requirements change volatility. On the other hand, Valente et al. [19] present correlations between degree, betweenness, closeness, eigenvector centrality measures, and indicate that those measures are distinct but notionally related. Thus in this work instead of employing 40 metrics, we chose the metrics suggested in [19] to predict requirements volatility for AVPRJ. These centrality metrics give each software requirement a value regarding their position in network. Brief explanations of the employed network metrics are given in Table 4.

Hein et al. [12] used language processing to create network for requirements. In this study instead we used traceability information to create network graph for software requirements. Traceability links from software requirements to system requirements are used for this purpose. We assigned weights between software requirements regarding system requirement traceability links. Software requirements which are derived from similar system requirements are tend to be closer in our model. Weight assignment formula is given below (Equation 1). W is weight between software requirements, NCLINK is the number of common system requirement links between two software requirements and NTOTLINK is the total number of system requirements linked from those two software requirements. After weight assignment, a symmetrical $n \times n$ matrix is created where n denotes the number of software requirements. Then the network metrics are computed over this matrix.

$$W_{ij} = \frac{NCLINK_{i,j}}{NTOTLINK_{i,j}} \tag{1}$$

3.4. Model Output

Our proposed model output is the number of change requests per software requirement. After the SRS document is reviewed and completed for AVPRJ, change requests linked to each software requirement are reported in the issue management system, and the document is modified accordingly by the analysis team. Thus we define requirements volatility in our industrial context with respect to number of change requests that have been applied to add a new requirement or to modify an existing requirement in the associated SRS document. Please note that our model outputs decimal values, but number of change request per requirement in practice can only get integer values. Therefore we round fractional parts to the nearest integer.

3.5. Tools

We wrote scripts to extract requirement quality and project specific metrics from SRS documents. Later, UCINET tool [20] is used to create network metrics from the matrix that we extracted based on software and system requirements. Regression models with different machine learners are trained using WEKA tool [21]. Prediction results are further post-processed in MATLAB to obtain the performance measures regarding all RQs.

3.6. Machine Learning Techniques

We train models using linear regression, random forest regression, support vector regression and k-nearest neighbor regression methods. Linear regression was utilized in [5], whereas classifier version of the other three techniques were used in [12].

For k-nearest neighbor regression, inversely proportional weighting option is selected. Higher weights are assigned to closer training samples which resulted in better prediction results for our model. For support vector

regression commonly used radial basis function kernel is selected. Increasing gamma parameter too much may result in over-fitting [22] and we also experienced a great computational cost with little to no prediction success gain for large gamma. Thus C and gamma parameters are assigned as 1.

In this study 10-fold cross validation technique is used to split training and test sets. Firstly, the dataset containing all software requirements is shuffled randomly and split into 10 groups of approximately equal size. One group is labeled as a test set and other groups are used to train machine learning models. This procedure is repeated 10 times until each unique group is used as test set once.

3.7. Performance Evaluation

For RQ 1, the following measures are used for performance evaluation: Mean Magnitude of Relative Error (MMRE), Median Magnitude of Relative Error (MdMRE), Pred(0.5) and Pred(0.25) [23]. Relative error is calculated according to Equation 2. $Err_{relative}$ is relative error, Val_{act} is the actual value, whereas Val_{pred} is the predicted value.

$$Err_{relative} = \frac{|Val_{act} - Val_{pred}|}{|Val_{act}|} \quad (2)$$

There are requirements with zero change requests. Thus division by zero problem arises while calculating relative error. We made an assumption for unchanged requirements as presented in Equation 3.

$$\text{If } Val_{act} = 0 \qquad Err_{relative} = \frac{|Val_{act} - Val_{pred}|}{1} \quad (3)$$

Pred(k) is a measure of variance of the error distribution. This measure is based on relative error and it shows the percentage of predictions whose errors are less than or equal to k.

For RQ2, we aim to predict highly volatile requirements, and thus, we first employ a method to identify those among the set of requirements:

- Step 1: Rank requirements by their actual number of change requests in descending order and record their rank as R_{actual}.
- Step 2: Obtain regression prediction results for each software requirement.
- Step 3: Rank requirements by their predicted number of change requests in descending order and record their rank as $R_{predicted}$.
- Step 4: Evaluate results according to the listing in Table 5. P denotes percentage of requirements which are perceived as highly volatile, and N_{req} denotes total number of requirements in validation set.

Table 5
Requirements volatility rank results evaluation

Condition	Evaluation
R_{actual},$R_{predicted} \leq N_{req} \times P$	True Positive
R_{actual},$R_{predicted} > N_{req} \times P$	True Negative
$R_{actual} \leq N_{req} \times P$, $R_{predicted} > N_{req} \times P$	False Negative
$R_{actual} > N_{req} \times P$, $R_{predicted} \leq N_{req} \times P$	False Positive

- Step 5: Calculate recall, accuracy and false alarm rate.

Table 5 can be interpreted as follows: True Positive instances are requirements that are actually highly volatile and the model also categorizes those as highly volatile. In the case of True Negatives, a requirement is actually less volatile, so is its prediction. False Negatives occur when highly volatile requirements are regarded as less volatile by the predictor. Finally, False Positives indicate less volatile requirements predicted as highly volatile.

To answer RQ2, recall, accuracy and false alarm rate measures are computed. Recall result shows how successful model in predicting highly volatile requirements. According to us this measure is the most important one regarding RQ2. Accuracy measure shows the prediction success for both highly volatile and less volatile requirements. False alarm rate presents how much effort has put in vain by mis-evaluating less volatile requirements.

4. Results and Discussion

We present and discuss the performance of the models with respect to two RQs in this section. We also compare the performance of the prediction models proposed in this study with the prior work [5] .

4.1. RQ 1

After obtaining processed data, machine learning regression methods are applied to answer the question if requirement quality metrics, network metrics, project specific metrics can be used to predict the number of changes on each software requirement by employing machine learning methods. Model performance results are gathered for all input metric and machine learning method combinations separately.

Results for RQ 1 is given in Table 6. The following abbreviations are used: ML for machine learning, Q for requirement quality metrics, P for project specific metrics, N for network metrics, KNN for k-nearest neighbor regression, LR for linear regression, RF for random forest regression and SVR for support vector regression.

In terms of input metric combinations, the best MMRE results are achieved with Q&P&N(0.366), Q&N(0.381) and

Table 6
Performance evaluation results for RQ 1

Metrics+ML method	MMRE	MdMRE	Pred(0.5)	Pred(0.25)
Q&P&N+KNN	0.366	0	0.681	0.57
Q&P&N+LR	0.53	0.5	0.524	0.411
Q&P&N+RF	0.392	0	0.663	0.545
Q&P&N+SVR	0.392	0	0.662	0.554
Q&P+KNN	0.402	0	0.641	0.529
Q&P+LR	0.513	0.5	0.541	0.428
Q&P+RF	0.45	0.333	0.6	0.486
Q&P+SVR	0.459	0.5	0.584	0.479
P&N+KNN	0.422	0	0.632	0.52
P&N+LR	0.55	0.667	0.484	0.372
P&N+RF	0.454	0.5	0.595	0.48
P&N+SVR	0.469	0.5	0.561	0.475
Q&N+KNN	0.381	0	0.665	0.553
Q&N+LR	0.534	0.5	0.52	0.407
Q&N+RF	0.394	0	0.662	0.545
Q&N+SVR	0.426	0	0.621	0.515
Q+KNN	0.443	0.333	0.598	0.488
Q+LR	0.512	0.5	0.542	0.43
Q+RF	0.455	0.5	0.594	0.483
Q+SVR	0.483	0.5	0.556	0.446
P+KNN	0.555	0.667	0.483	0.37
P+LR	0.548	0.667	0.485	0.373
P+RF	0.556	0.667	0.483	0.371
P+SVR	0.516	0.5	0.512	0.417
N+KNN	0.448	0.5	0.596	0.482
N+LR	0.549	0.667	0.484	0.372
N+RF	0.485	0.5	0.561	0.446
N+SVR	0.53	0.5	0.5	0.392

Table 7
Comparison of our performance (RQ 1) against [5]

	MMRE	MdMRE	Pred(0.25)	Pred(0.5)
Q+LR	0.51	0.5	0.43	0.54
Best model	0.36	0	0.57	0.68
NLines+LR [5]	0.58	0.27	0.5	0.63

Q&P(0.402). We may interpret that requirement quality metrics (Q) are successful at predicting number of change requests per software requirement, and its combinations with the other metrics also give good results. With respect to the machine learning algorithm, the three best performing metric combinations give the highest prediction performance when k-nearest neighbor algorithm is utilized.

MdMRE is zero for the following metric and machine learner combinations: Q&P&N+KNN, Q&P&N+RF, Q&P&N+SVR, Q&P+KNN, P&N+KNN, Q&N+KNN, Q&N+RF and Q&N+SVR. Number of change requests for more than half of the software requirements are predicted correctly with these models. Since many prediction models give the same best result, we do not rank the best performing models with regard to MdMRE.

The best performance with respect to Pred(0.25) and Pred(0.5) are obtained with Q&P&N+KNN. Q&N+KNN and Q&P+KNN report the second and third best results. Those results indicate that requirement quality metrics and k-nearest neighbor algorithm are also successful with respect to Pred measures.

To sum up, best performance results are achieved by using requirement quality metrics, project specific metrics and network metrics altogether. Accordingly, K-nearest neighbor algorithm gives best performance results for all measures. In all best performing models, quality metrics are utilized either as a pair with project or network metrics or as combination of all three. It seems the way requirements are documented has a high effect on the volatility rates.

We compared our findings against the study conducted by Loconsole et al. [5]. Table 7 reports the performance of linear regression model with the best metric set in our study, our best performing model and the best performing model of [5]. If we compare the findings only on LR, we observe that using number of lines predicts volatility better on their commercial setting, while in our context using quality metrics only does not give the best result. Other algorithms like KNN in combination with all metrics significantly improve the prediction performance by reducing MMRE down to 0.36 and MdMRE down to 0, and increasing Pred(0.25) up to 57%.

4.2. RQ 2

RQ 2 aims to measure the success of our model in predicting highly volatile requirements. We present our technique to identify highly volatile requirements in Section 3.7. We first need to determine change request coverage to categorize highly-volatile requirements, and later calculate recall, accuracy and false alarm rates. Rates for various change request coverage by most volatile requirements are given in Table 8. As change request coverage grows more requirements are labeled as highly volatile. We chose 80% change request coverage since approximately 40% percent of reviewers are considered as well-experienced in AVPRJ. Therefore by applying this model we could assign review task of 38.6% of total requirements, which are possibly highly volatile, to experienced developers in early phase of development. We did not present other coverage results in this study due to page limitation.

In Table 9 the best recall results are achieved with Q&P&N+KNN(0.632), Q&N+RF(0.616) and Q&P+KNN(0.604). Again, all best performing models have requirement quality metrics in common, whereas the best combination consists of all metrics. The best accuracy results are obtained from Q&P&N+KNN(0.716),

Table 8
Change request and requirement coverage relation for AVPRJ

CR Coverage Percent	REQ Coverage
60%	20.5%
70%	29.6%
80%	38.6%
90%	47.7%

Table 9
Performance results of RQ 2 model for 80% CR coverage

Metrics+ML method	Recall	Accuracy	False Alarm Rate
Q&P&N+KNN	0.632	0.716	0.232
Q&P&N+LR	0.531	0.638	0.295
Q&P&N+RF	0.624	0.71	0.237
Q&P&N+SVR	0.591	0.684	0.258
Q&P+KNN	0.604	0.694	0.249
Q&P+LR	0.508	0.62	0.31
Q&P+RF	0.602	0.692	0.251
Q&P+SVR	0.558	0.658	0.278
P&N+KNN	0.574	0.671	0.268
P&N+LR	0.418	0.55	0.366
P&N+RF	0.557	0.658	0.279
P&N+SVR	0.496	0.61	0.318
Q&N+KNN	0.614	0.702	0.243
Q&N+LR	0.53	0.637	0.296
Q&N+RF	0.616	0.703	0.242
Q&N+SVR	0.569	0.667	0.271
Q+KNN	0.586	0.68	0.261
Q+LR	0.508	0.62	0.31
Q+RF	0.582	0.677	0.263
Q+SVR	0.529	0.636	0.296
P+KNN	0.503	0.616	0.313
P+LR	0.423	0.554	0.363
P+RF	0.496	0.611	0.317
P+SVR	0.462	0.584	0.339
N+KNN	0.557	0.657	0.279
N+LR	0.404	0.539	0.376
N+RF	0.565	0.664	0.274
N+SVR	0.498	0.612	0.316

Q&N+RF(0.703) and Q&P+KNN(0.694). The lowest false alarm rate results are achieved by Q&P&N+KNN(0.232), Q&N+RF(0.242) and Q&P+KNN(0.249). K-nearest neighbor and random forest regression methods are successful in predicting highly volatile requirements for 80% change request coverage.

The most important measure for RQ 2 is recall since the purpose of this question is to measure success on predicting highly volatile requirements. We correctly identify 63.2% of highly volatile requirements which are exposed to 80% of the total requirement changes.

4.3. Threats to Validity

Internal validity: In this study we present requirements volatility in a software project can be predicted to some extent utilizing requirement quality metrics, project specific factors and network metrics altogether. However, this does not imply causal relationship between input and output metrics since we did not conduct a controlled experiment.

External validity: We have conducted the case study on one project, so results have local validity. However, the dataset is quite large with more than 20,000 requirements from three distinct releases developed by many software developers. Nonetheless, applying the predictive models on different projects in the future would be better in terms of generalization of results.

Construct validity: Developers did not use their native language in software requirements. Thus there could be some typos which may affect textual requirement quality metrics. Also there could be some expressions used by developers in software requirements, e.g. subjective expressions that should have taken into consideration while creating requirement quality metrics but we missed. Due to the size of dataset we couldn't manually check these kinds of typos and grammatical errors, but we know that reviewers are responsible for correcting those. We create network graphs based on traceability links between software and system requirements as indicated in the SRS. We could have use linguistic data to connect software requirements as previous study [12] and it may reflect relationship between requirements in a better way. We plan to do it as a future work.

Conclusion validity: For RQ 2 only the results of 80% change request coverage are presented due to page limitation. Regarding the results of other CR coverage, we observe higher recall and accuracy whereas false alarm rate grows undesirably as the coverage grows. Therefore RQ 2 results would differ in that way if we had chosen other CR coverage rate.

5. Conclusion and Future Work

In this paper, we have carried out an empirical study to predict number of changes per software requirement by using requirement quality measures, project specific factors and requirement interdependencies. 22,771 software requirements from a safety-critical software project in ASELSAN are utilized to build 28 prediction models and assess the best performing metric suite and algorithm. We conclude that we can predict volatility of requirements with an average MMRE of 36% by observing metrics of similar requirements through KNN. We also observe that measuring requirements from different aspects like quality, project and network dependencies gives a much better performance. We plan to integrate

such a predictor model into requirement management tools like DOORS to be used prior to the SRS review activity so that highly-volatile requirements could be automatically and accurately identified. This way, software development leads could take precautions beforehand to reduce requirements volatility related risks. Since there is not enough empirical studies conducted in related area, more empirical research should be carried out to validate the best performing models.

References

[1] N. Nurmuliani, D. Zowghi, S. Powell, Analysis of requirements volatility during software development life cycle, in: 2004 Australian Software Engineering Conference, ASWEC '04, IEEE, 2004, pp. 28–37.

[2] G. Swathi, A. Jagan, C. Prasad, Writing software requirements specification quality requirements: An approach to manage requirements volatility, Int. J. Comp. Tech. Appl. 2 (2011) 631–638.

[3] R. Thakurta, A mixed mode analysis of the impact of requirement volatility on software project success, Journal of International Technology and Information Management 20 (2011).

[4] A. M. Alsalemi, E.-T. Yeoh, A systematic literature review of requirements volatility prediction, in: 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication, ICCTCEEC-2017, IEEE, 2017, pp. 55–64.

[5] A. Loconsole, J. Börstler, Construction and Validation of Prediction Models for Number of Changes to Requirements, Umeå University Technical Report UMINF-07.03, Umeå University Department of Computing Science, UMEÅ, SWEDEN, 2007.

[6] D. F. X. Christopher, E. Chandra, Prediction of software requirements stability based on complexity point measurement using multi-criteria fuzzy approach, International Journal of Software Engineering & Applications 3 (2012) 101–115.

[7] L. Shi, Q. Wang, M. Li, Learning from evolution history to predict future requirement changes, in: 2013 21st IEEE International Requirements Engineering Conference, RE-2013, IEEE, 2013, pp. 135–144.

[8] A. Goknil, R. van Domburg, I. Kurtev, K. van den Berg, F. Wijnhoven, Experimental evaluation of a tool for change impact prediction in requirements models: Design, results, and lessons learned, in: 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), MoDRE 2014, IEEE, Karlskrona, Sweden, 2014, pp. 57–66.

[9] B. Morkos, J. Mathieson, J. D. Summers, Comparative analysis of requirements change prediction models: manual, linguistic, and neural network, Research in Engineering Design 25 (2014) 139–156.

[10] X. Wang, C. Wu, L. Ma, Software project schedule variance prediction using bayesian network, in: 2010 IEEE International Conference on Advanced Management Science, volume 2 of *ICAMS 2010*, IEEE, Chengdu, China, 2010, pp. 26–30.

[11] T. Nakatani, T. Tsumaki, Predicting requirements changes by focusing on the social relations, in: Proceedings of the Tenth Asia-Pacific Conference on Conceptual Modelling, volume 154 of *APCCM 2014*, Auckland, New Zealand, 2014, pp. 65–70.

[12] P. H. Hein, E. Kames, C. Chen, B. Morkos, Employing machine learning techniques to assess requirement change volatility, Research in Engineering Design 32 (2021) 245–269.

[13] W. Pedrycz, J. Iljazi, A. Sillitti, G. Succi, Prediction of the successful completion of requirements in software development—an initial study, in: Agent and Multi-Agent Systems: Technology and Applications, Springer, 2016, pp. 261–269.

[14] T. J. Ostrand, E. J. Weyuker, How to measure success of fault prediction models, in: Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, SOQUA'07, Dubrovnik, Croatia, 2007, pp. 25–30.

[15] T. Clancy, The standish group report, Chaos report (1995).

[16] Y. Jiang, B. Cukic, T. Menzies, Fault prediction using early lifecycle data, in: The 18th IEEE International Symposium on Software Reliability, ISSRE'07, IEEE, 2007, pp. 237–246.

[17] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, V. Moreno, A framework to measure and improve the quality of textual requirements, Requirements engineering 18 (2013) 25–41.

[18] A. Faisandier, Systems architecture and design, Sinergy'Com Belberaud, France, 2013.

[19] T. W. Valente, K. Coronges, C. Lakon, E. Costenbader, How correlated are network centrality measures?, Connections 28 (2008) 16–26.

[20] S. P. Borgatti, M. G. Everett, L. C. Freeman, Ucinet for windows: Software for social network analysis, Harvard, MA: analytic technologies 6 (2002).

[21] F. Eibe, M. A. Hall, I. H. Witten, The weka workbench. online appendix for data mining: practical machine learning tools and techniques, in: Morgan Kaufmann, 2016.

[22] A. Ben-Hur, J. Weston, A user's guide to support vector machines, in: Data mining techniques for the life sciences, Springer, 2010, pp. 223–239.

[23] D. Zhang, J. J. Tsai, Machine learning applications in software engineering, volume 16, World Scientific, 2005.