Learning from Evolution History to Predict Future Requirement Changes

Lin Shi[1,2], Qing Wang[1,3], Mingshu Li[1,3]
[1]Laboratory for Internet Software Technologies, Institute of Software Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China
{shilin,wq}@itechs.iscas.ac.cn, mingshu@iscas.ac.cn

***Abstract*—Managing the costs and risks of evolution is a challenging problem in the RE community. The challenge lies in the difficulty of analyzing and assessing the proneness to requirement changes across multiple versions, especially when the scale of requirements is large. In this paper, we define a series of metrics to characterize historic evolution information, and propose a novel method for predicting requirements that are likely to evolve in the future based on the metrics. We apply the prediction method to analyze the product updates history through a case study. The empirical results show that this method can provide a tradeoff solution that narrows down the scope of change analysis to a small set of requirements, but it still can retrieve nearly half of the future changes. The results indicate that the defined metrics are sensitive to the history of requirements evolution, and the prediction method can reach a valuable outcome for requirement engineers to balance their workload and risks.**

***Index Terms*—Requirements evolution, prediction model, volatility measure, case study.**

I. Introduction

Most software systems tend to iteratively evolve and have a large number of versions [11][19]. With the increasing complexity and scale, volatile requirements could lead to risks on the costs and schedule overruns, or even worse, project failures [14]. During long lifetimes, certain requirements are frequently changed across versions because of a variety of factors such as immaturity business processes, multiple dependencies with other requirements, and inflexible designs to adapt changes [24].

Being aware of volatile requirements can contribute to efficient requirements evolution management. First, requirements analysts could reinforce their communications with customers by paying more attentions on volatile requirements. They could try to inspire customers to elicit more in-depth needs. It is significant to identify potential changes in elicitation stage rather than allowing them to be delayed into late production stages, since making changes in the production stage could cost 100 times more than making the same changes during the early requirements and design stages [5]. Second, identifying volatile requirements is important to make designs more robust to adapt future evolutions. For example, architects could optimize their designs by encapsulate and isolate volatile requirements [13]. They could also make designs that deliver given functionality in many different ways rather than delivering in only a single way.

Existing literatures on requirements evolution prediction focuses mainly on analyzing possible future volatilities [6][7] and predicting evolution characteristics of the overall requirements [20][21]. Studies that can predict which requirements have the potential risks of being changed are rarely exploited. It is challenging to conduct such a study because the predicting factors of evolution are difficult to comprehend and construct.

In this paper, we propose a methodology to predict requirements that are possible to be changed in the future based on historic information. In this methodology, we present metrics that we define to measure evolution, the procedure of training and predicting, and the measurement of predicting performance. We also provide a solution to balance the workload of volatility analysis and the precision of predicting changes. Based on the methodology, we conduct a case study on an industrial software product, which has 13 historic versions that involve 4,044 requirements and 800 requirement changes in total. The results show that the prediction method can narrow down the scope of the volatility analysis while keeping the precision of predicting the results at a certain level. We believe that the prediction model can help practitioners to better control the requirements evolution in practice.

The main contributions of the paper are as follows:

- We define a set of metrics to measure the evolution history of a requirement.
- We propose a novel solution for reducing the impact of the requirements evolution by predicting requirements that are prone to change in the future.
- Based on our methodology, we build a prediction model in a real industrial case study, and evaluate the model from its quantitative aspects. The results show that the model is efficient and allows practitioners to better control the impact of the requirements evolution.

The rest of this paper is organized as follows: Section II describes the methodology of our study. Section III presents the case study we conduct in a real industrial setting. Section IV discusses the issues and future work of our study. Section V presents threats to the validity. Section VI introduces related work, and Section VII concludes the paper.

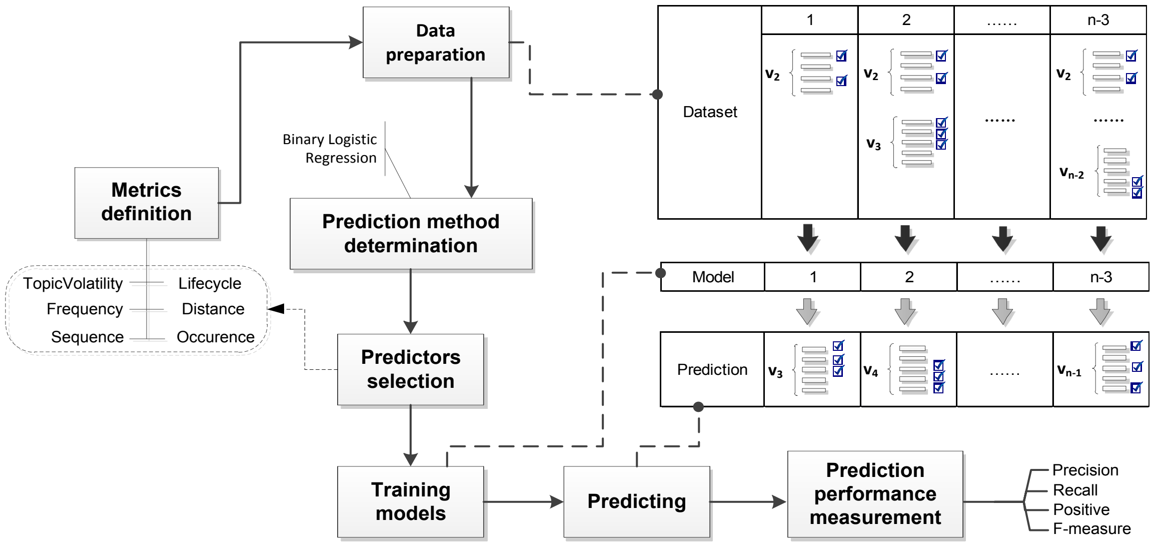

Figure 1. The procedure of predicting requirements evolution

II. METHODOLOGY

Our main research goal is to determine how well we can predict future requirement changes based on information from the evolution history. We define requirements evolution as additions, modifications, and deletions. In this study, we are concerned with newly added and modified requirements because deleted requirements are typically meaningless for predicting the future.

We model the probability of requirements evolution based on the following hypotheses.

- Requirements that are frequently changed in historic versions are also likely to change in the future.
- Requirements that belong to topics that are prone to change in historic versions are also prone to change in the future.

Figure 1 shows the procedure of prediction in this study. First, we define a number of metrics to characterize the evolution history of a product with multiple versions. Then, we prepare datasets to be used in training models and predicting. We use different datasets for building and evaluating the prediction models. For a product with *n* historic versions, our intention is to discover the set of volatile requirements that could change in the unknown version *n+1*. Because change labels of requirements in version *n* cannot be obtained until the *n+1* version has been delivered or released, we use historic evolution information from *n-1* versions to build the prediction models, and we evaluate the models by comparing the predicted results to the actual changes in the requirements, as in the example shown in Figure 1. In addition, if historic information of early versions is not sufficient to build prediction models, then we can filter out those early versions and start from later versions.

As shown in the tables of Figure 1, for a product with *n* versions, there are *n-1* versions that can be used to evaluate the prediction models trained by earlier versions. Therefore, we can evaluate at most *n-2* versions. We cannot build a prediction model based on the first version because all of the requirements in the first version are newly added with no historic information, and the metrics of the evolution history thus cannot work. Thus, excluding the first version, we can build at most *n-3* testable prediction models that can be evaluated by actual change results. After that, we determine the prediction method and selected predictors using the defined metrics. We train models from a series of history versions. Finally, we measure the performances of the prediction models using a variety of indicators.

A. Data Preparation

By iteratively comparing the requirements of two adjacent versions, we can identify the evolution history of each requirement, including its additions, modifications, and deletions during historic versions. We record such information into an *evolution matrix*, as shown in TABLE I. We assign a value of 2 to the newly added requirements, a value of 1 to the modified requirements, a value of -1 to the deleted requirements, and a value of 0 to the unchanged requirements respectively. For example, consider the requirement that Req.ID *r* has been added in version 3, changed in version 6, and continuously changed in version 7. Version 1 is not shown in TABLE I, because it is the initial version, and the requirements in that version are all newly added.

Based on the evolution matrix, we can define and calculate metrics that characterize the evolution history of a requirement conveniently.

TABLE I THE EVOLUTION MATRIX FOR REQUIREMENTS

(Where added = 2, modified = 1, deleted = -1, unchanged = 0, number of historic versions = 8)

Version / **Req. ID**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	0	2	1	0	-1	0
…	…	…	…	…	…	…	…
r	0	2	0	0	1	1	0

B. Metrics Definition

We are interested in metrics that can reflect the following aspects of requirement changes.

- The metric should indicate the volatility of the topic that a requirement belongs to.
- The metric should measure how frequently a requirement changes in its historic versions.
- The metric should describe the length of the time interval between the version in which a requirement is added or changed to the latest version.

Accordingly, we construct 6 metrics as shown in TABLE II, and the content-related metric is *topicVolatility*. Metrics that measure the degree of change include *frequency* and *sequence*. Metrics that reflect the length of change time interval include *distance* and *lifecycle*. We also design a combination metric *occurrence* to present the intensive changing position of a change. In this section, we will describe these metrics in more detail.

TABLE II THE METRICS FOR REQUIREMENTS EVOLUTION

Category	Metric names	Description
Content	TopicVolatility (TV)	Average percentage of changed requirements over all requirements on the same topic per version
Change degree	Frequency (F)	Sum of the times of changes for a requirement over all historic versions
	Sequence (S)	Maximum number of versions that a requirement continuously changed
Length of change time	Distance (D)	Sum of the number of versions between the version when a requirement changed and the latest version.
	Lifecycle (LC)	The number of versions between the version that a requirement is added and the latest version
Combinati on	Occurrence (OC)	A composed metric of *Frequency*, *Distance*, and *Lifecycle*. The metric is designed to reflect the intensively changing position of a requirement toward its lifecycle.

TopicVolatility. For software systems that have a large set of features, the requirements are typically organized into topic-wise groups. For example, practitioners are likely to group requirements that are derived from relevant and coherent topics into the same groups when building features [27] and feature trees [9] for release planning.

We define this metric here to measure the topic-volatility of a requirement. Thus, to testify to our intuition, whether requirements belong to topics that are prone to change in historic versions is also likely to change in the future. We assume that the topic of a requirement can be directly extracted from the requirements documentation or implicitly discovered by text mining techniques. For a requirement *r*, if *r* belongs to topic *t*, then the *topicVolatility* is

$$TopicVolatility\ (r) = \frac{1}{n-2}\sum_{i=2}^{n-1}\frac{|R_C(i,t)|}{|R(i,t)|} \tag{1}$$

Where i denotes the version number, n denotes the total number of versions, t denotes the topic t, $R_C(i,t)$ denotes the requirements of version i under the topic t that have been changed in version i+1, R(i,t) denotes the requirements of version i under topic t. We exclude the version n because $R_C(n,t)$ is typically unavailable unless the version n+1is planned.

Frequency measures the total times that a requirement has been changed in all of its historic versions. We divide the number of times by *n-1* to scale the value between 0 and 1. For example, in TABLE I, the requirement *r* has been added in version 3, and modified in version 6 and 7; thus, *Frequency (r)* = (1+1+1)/7 = 0.43.

$$Frequency\ (r) = \frac{1}{n-1}\sum_{i=2}^{n} isChanged(r,i) \tag{2}$$

Where i denotes the version number, n denotes the total number of versions, r denotes the given requirement. isChanged(r,i) is 1 if r have been changed in version i, 0 otherwise.

Sequence measures how long a requirement continuously changed in all of its historic versions. This metric reflects the degree of requirement change from the activity aspects. Unstable requirements can continuously change during the evolution history. For example, in TABLE I, the requirement *r* has been continuously changed in version 6 and 7; thus, *Sequence(r) = 2*. There could be some interesting causes that underlie a high value of the sequence for a requirement. For example, the requirements that have been continuously changed in historic versions could be caused by inflexible designs, defects from previous versions [23], and difficult-to-satisfy business processes.

$$Sequence\ (r) = max(N(r,i)),\ (2 \leqslant i \leqslant n) \tag{3}$$

Where r denotes the given requirement, n denotes the total number of versions, i denotes the version number. N(r,i) denotes the number of versions that r continuously changed from its added version to the version i.

Distance is the sum of all of the distances from the current version number to all of the version numbers. Considering the version number to be ages of the product, this metric reflects the total age of the changes for a requirement. For example, in TABLE I, the requirement *r* has been added in version 3 and modified in versions 6 and 7; thus, *Distance(r)* = (8-3)+(8-6)+(8-7) = 8.

$$Distance\ (r) = \sum_{i=2}^{n}(n-i) \tag{4}$$

Where r denotes the given requirements, i denotes the version number in which the requirement r has been changed, n denotes the total number of versions.

Lifecycle. We design this metric to testify to our intuition that whether requirements with short lifetimes are unstable. For a specific requirement *r*, the measurement can reflect the lifetimes from the first version that it was added to, to the latest version *n*. For example, requirement *r* was added in version 3; when considering the lifecycle of the requirement in version 8, the *Lifecycle(r)*=8-3+1=6.

$$Lifecycle\ (r) = n - VA + 1 \tag{5}$$

Where n denotes the total number of versions, VA denotes the version number that requirement r was newly added.

Occurrence. This metric synthetically reflects in which position a requirement is intensively changed during its lifecycle. We divide *Distance(r)* by *Frequency(r)* to calculate the average lifetime of the changes of a requirement. For example, for requirement *r*, *Occurrence* (*r*) = 8/((3/7)*6) = 3.11.

$$Occurrence\ (r) = \frac{1}{Lifecycle\ (r)} * \frac{Distance\ (r)}{Frequency\ (r)} \qquad (6)$$

Where r denotes the given requirement, Distance(r), Frequency(r), and Lifecycle(r) are calculated from Equation (4), Equation (2), and Equation (5).

C. Prediction Method and Predictor Selection

In this study, we select logistic regression to be the prediction method for the following reasons. Logistic regression is a prediction approach that can be used when the target variable is a categorical variable with two categories. Logistic regression is different from other regression techniques, such as linear regression, and aims to determine whether there is some form of functional dependency between the explanatory variables and the dependent variable. Logistic regression does not assume a linear relationship between the explanatory variables and the probability function. It does not assume homoscedasticity. Instead, logistic regression equations are solved iteratively, and the maximum likelihood method is typically used to estimate parameters in logistic regression equations.

In our case, the target variable indicates whether a requirement will change in the next version or not. The target variable is one if a requirement changed in the next version, and zero otherwise.

$$Logit\ (P(change)) = w_1*\alpha_1 + w_2*\alpha_2 + \ldots\ldots + w_n*\alpha_n + w_0. \qquad (7)$$

Where $Logit(p) = \log\{p/(1-p)\}$*,* α_i *denotes the predictors selected in the model,* w_0 *is the intercept.*

The logistic regression provides the probability of change for a requirement. We need to determine a cutoff point to distinguish the probability between 0 and 1. An operable process for selecting cutoff point will be introduced in Section III.C.

We have defined a series of metrics in Section II.B to characterize the evolution history of the requirement changes. We try to use metrics that behave well in most datasets as predictors of the future versions. We use the model selection technique, stepwise regression [8], to select the best metrics on individual datasets. This technique is conducted by iteratively deleting predictors from the full model until deleting any remaining predictor would provide no further improvement. In addition, the *occurrence* metric is a function of *frequency*, *lifecycle*, and *distance*. In statistics, independent predictors should be selected for logistic regression equations, since they can yield a more stable solution. However, if a combination metric can contribute to effective prediction models, then it is also a reasonable option to consider it as an explanatory variable [22].

After conducting a stepwise regression, we can obtain the sets of good predictors for each historic version. We try to choose predictors that perform well in overall history versions to construct a prediction model for future changes. Then we use the median of the model numbers as the boundary. If there are *n* best models that were trained by stepwise regression, then we count the total number of times that a metric appeared in *n* models. We select those metrics that appeared more than *n/2* times to be predictors in the general prediction model.

D. Training and Predicting

We construct the datasets used in training regression models by two steps.

First, we obtain the values of metrics we defined for requirements of version *2* to v_{n-1}. An example of the dataset used for training is shown in TABLE III. Each item contains the identifier of a requirement, values of the 6 metrics, and a change label that can reflect whether the requirement have been changed in the next version.

TABLE III AN EXAMPLE OF THE DATASET FOR TRAINING AND PREDICTING

Req.ID	TV	F	S	D	LC	OC	*Change*
1	0.24	0.25	0.33	3	5	2.4	0
2	0.44	0.5	0.33	3	4	1.5	1
3	0.36	0.75	0.18	4	5	1.1	1

Second, to train the models that are applicable for most of the evolutions, we construct the training datasets by assembling sets of historic requirements data. For example, the training set for version 3 contains all of the requirements from version 3 and the requirements from version 2 as well.

To evaluate the prediction power under realistic conditions, we use the trained model to predict the requirements evolution for the next version.

E. Prediction Performance Measurement

The performance of two-category prediction models is typically evaluated by a confusion matrix, as shown in TABLE IV. In this study, we use the commonly used prediction performance measures: *precision*, *recall*, *positive* and *F-Measure* to evaluate the models. These measures are derived from the confusion matrix.

TABLE IV A CONFUSION MATRIX

		Predicted	
		Not change	***change***
Actual	***Not change***	TN = True Negative	FP = False Positive
	change	FN = False Negative	TP = True Positive

Precision is defined as the ratio of the number of requirements correctly predicted to be changed to the total number of requirements predicted to be changed.

$$Precision = \frac{TP}{TP+FP}$$

Recall is defined as the ratio of the number of requirements correctly predicted to be changed to the total number of requirements that are actually changed.

$$Recall = \frac{TP}{TP+FN}$$

Positive is defined as the proportion of the number of requirements predicted to be changed to the total number of requirements. This measure can reflect the workload of further analysis on possible changing requirements, and it will be used to determine the cutoff point.

$$Positive = \frac{TP+FP}{TP+FN+TN+FP}$$

F-Measure is the harmonic mean of the *precision* and *recall*. It is a measure to the test's accuracy in statistics. It has been widely used in evaluating results of retrieval results [30]. In our study, we use this measure to evaluate the prediction results and consider both the *precision* and *recall* to be equally important.

$$F\text{-}Measure = \frac{2*Precision*Recall}{Precision+Recall}$$

III. Case Study

To evaluate the prediction method we presented in Section II, we conduct a case study using 13 historic versions of requirements from an industrial product. In this section, we present the predicting process and results.

A. Data Sources

This case study is conducted on the company of National Fundamental Software (NFS) of China Ltd[1]. We collected requirements and historic evolution information from an eight-year product, which is a software system supports for enterprise software process management [2]. The product provides a cooperative work platform that covers most of the lifecycle of software development. It has 13 main versions from 2004 to 2012 and is written in Java. The average size of the product is 1,200 KLOC.

TABLE V Characteristics of the dataset

#version	1	2	3	4	5	6	7	8	9	10	11	12	13
Release time	2004	2005	2006	2007	2007	2008	2009	2009	2010	2010	2011	2011	2012
# req.	186	189	195	221	276	310	308	334	355	389	394	426	461
	A	3	6	26	60	52	11	38	21	34	5	32	35
	M	10	22	60	43	57	26	28	36	77	20	28	32
	D	0	0	0	5	18	13	2	0	0	0	0	0

TABLE V summaries some of the main characteristics of the product evolution history. The requirements of the product are presented in the format of textual use cases. Row "A" represents the number of added requirements during each updates. Row "M" represents the number of modified requirements during each of the updates. Row "D" represents the number of deleted requirements during each update. In total, we have collected data for 4,044 requirements with 800 changes in total.

We collected requirement documents and change logs about the 13 versions from the product's repository. The requirements are presented by textual use cases writing in natural language. By comparing the requirement documents of two contiguous versions, we can identify added, deleted, and modified requirements during each update. The comparison is automatically conducted by text comparison tools. When the requirements are not presented by the text, we refer to the change logs to identify the changes. The identified changes are confirmed by practitioners from the case company. With that knowledge, we can automatically generate the *evolution matrix* introduced in section II.A for each version.

Moreover, the requirements of the product are organized into topic-wise groups. We can conveniently obtain the topic and calculate the value of *topicVolatility* for each requirement. We implemented a tool to automatically calculate the values of the metrics based on the evolution matrix.

Then, we formed sets of requirements with the values of *topicVolatility, frequency, sequence, distance, lifecycle, occurrence*, and labels of the actual change results for each history version, which are basic data of the datasets for training and predicting. We constructed datasets that were used for training and testing as shown in TABLE VI.

TABLE VI Datasets Construction

#	datasets for training	datasets for testing
1	$\{v_3\}$	$\{v_4\}$
2	$\{v_3, v_4\}$	$\{v_5\}$
3	$\{v_3, v_4, v_5\}$	$\{v_6\}$
……	……	……
9	$\{v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}\}$	$\{v_{12}\}$

B. Predictor Selection and Building Models

As introduced in Section II.C, we used SPSS[3] to run stepwise regression on the 9 datasets listed in TABLE VI. When conducting the stepwise regression, we selected the "backward" strategy provided by SPSS, to include more predictors. We can obtain 9 sets of good predictors. Predictors that are selected for predicting the future version are recorded in TABLE VII.

TABLE VII Metrics selected by SPSS stepward regression

# Regression	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
TopicVolatility					√	√	√	√	√
Frequency					√	√	√	√	√
Distance			√						
Lifecycle	√	√					√	√	√
Occurrence			√	√	√	√	√	√	√
Sequence	√	√	√	√					

Basically, the metrics we defined are used to characterize evolution history. They tend to be more sensitive and representative of an evolution with a long history. We can observe that some predictors perform well for early versions, while others are more suitable for versions with a long history. We consider selecting predictors that frequently perform well in most of the models. We count the number of total times that one metric appears in the 9 models. The result is shown in

[1] http://www.nfschina.com/

[2] http://qone.nfschina.com/en/

[3] http://www.ibm.com/software/analytics/spss/

Figure 2. We select metrics that appear more than the median (9/2 = 4.5) times as a boundary for the good predictors. The final selected predictors are *topicVolatility*, *frequency*, *lifecycle*, and *occurrence*, as shown in TABLE VII, which are also the good predictors for the last three versions.

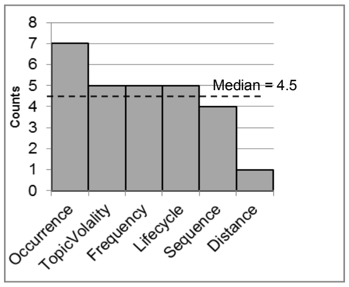

Figure 2. The predictor selection results

We use those selected predictors to build a prediction model, as follows:

$$\text{Logit}(P(\text{change})) = w_1 * \text{topicVolatility} + w_2 * \text{frequence} + w_3 * \text{lifecycle} + w_4 * \text{occurrence} + w_0, \quad (8)$$
$$\text{where } \text{Logit}(p) = \log\{p/(1-p)\},\ w_0 \text{ is the intercept.}$$

For each dataset, we build a corresponding model using equation (8). The regression results from SPSS are shown in TABLE VIII. The results show that 8 prediction models are significant at the level of 0.05. Only the significance of model 2 is larger than 0.05. For this model, we cannot reject the null hypothesis that there is no relationship between the set of predictors and the target variable. The first 3 models had predictors with a significance larger than 0.05, which means that those predictors are not significantly different from zero. However, models trained with datasets that have at least 6 versions are significant in both the model and predictor coefficients. The results show that the predictors we select are significant in most of the models.

TABLE VIII COEFFICIENTS FOR THE PREDICTION MODELS

#Model	P-value	TopicVolatility		Frequency		Lifecycle		Occurrence	
		B	*Sig.*	*B*	*Sig.*	*B*	*Sig.*	*B*	*Sig.*
1	0.001	7.008	0.004	2.317	0.022	1.070	0.032	-1.430	**0.152***
2	**0.357***	-0.466	**0.687***	1.024	**0.083***	0.189	**0.168***	-0.327	**0.411***
3	0.000	-0.570	0.516*	1.225	0.010	0.238	0.001	-0.737	0.001
4	0.000	-1.656	0.042	1.338	0.003	0.155	0.004	-0.774	0.000
5	0.000	-1.898	0.017	1.518	0.000	0.134	0.003	-0.683	0.000
6	0.000	-2.448	0.001	1.463	0.000	0.101	0.005	-0.414	0.000
7	0.000	-2.060	0.004	1.070	0.003	0.117	0.000	-0.207	0.000
8	0.000	-1.925	0.007	1.297	0.000	0.110	0.000	-0.270	0.000
9	0.000	-1.751	0.012	1.323	0.000	0.090	0.000	-0.268	0.000

Topicvolatility, *frequency*, *lifecycle*, and *occurrence* are significantly different from zero in most of the models, which supports our main hypotheses that the properties of evolution history do affect the probability of change in the future, at least in the considered project. In addition, the selected predictors are also the good predictors for versions with longer history. To predict changes for early versions, other predictors shown in TABLE VII could perform well.

C. Prediction Results

In Section III.B, we build 8 significant models. In this section, we use these models to predict which requirements could change in the next version, and using actual changes to evaluate the prediction performance. We also present the process of how we choose cutoff points to balance the analysis workload and evolution risks.

We predict a requirement to be changed if its predicted probability is above a cutoff point. As mentioned in Section II.C, when we determine a cutoff point, we need make a balance between the *precision* and *recall*. To retrieve more requirements that actually change in the next version, we can lower the cutoff point, and then, more requirements with low probabilities are accounted for. However, reducing the cutoff point could also lead to a decrease in the *precision* of the prediction results, and thus, more effort would be applied to analyzing the requirements volatility. Choosing a cutoff point means attaining a tradeoff between the *recall* and *precision*:

- The higher the recall value, the greater control we have on the requirement changes.
- The higher the precision value, the more effective effort we spend on requirements volatility analysis.

We conduct a *cost-benefits* analysis by selecting the best cutoff point. The selection method is shown in Figure 3.

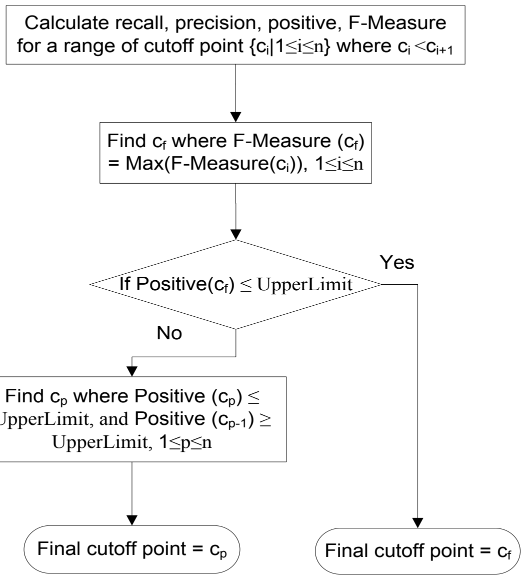
Calculate recall, precision, positive, F-Measure for a range of cutoff point $\{c_i|1\le i\le n\}$ where $c_i < c_{i+1}$
Find c_f where F-Measure (c_f) = Max(F-Measure(c_i)), $1\le i\le n$
If Positive(c_f) ≤ UpperLimit
Yes
No
Find c_p where Positive (c_p) ≤ UpperLimit, and Positive (c_{p-1}) ≥ UpperLimit, $1\le p\le n$
Final cutoff point = c_p
Final cutoff point = c_f

Figure 3. The cutoff point selection method

First, we need to select a range of cutoff points that covers most values of *precision* and *recall*. Figure 4 shows the values of the *recall*, *precision*, *positive* and *F-Measure* at different cutoff points.

Second, we determine the cutoff point (c_f) that has the highest F-measure. Because the *F-Measure* is the harmonic mean of the *precision* and *recall*. Since we consider precision and recall to be equally important, the cutoff point c_f represents the best balance point for the *recall* and *precision*. The dashed lines in Figure 4 denote *F-Measure* values, and we highlight the c_f in a circle for each model. We attempt to simply choose the c_f as the final balanced cutoff point. However, we find that

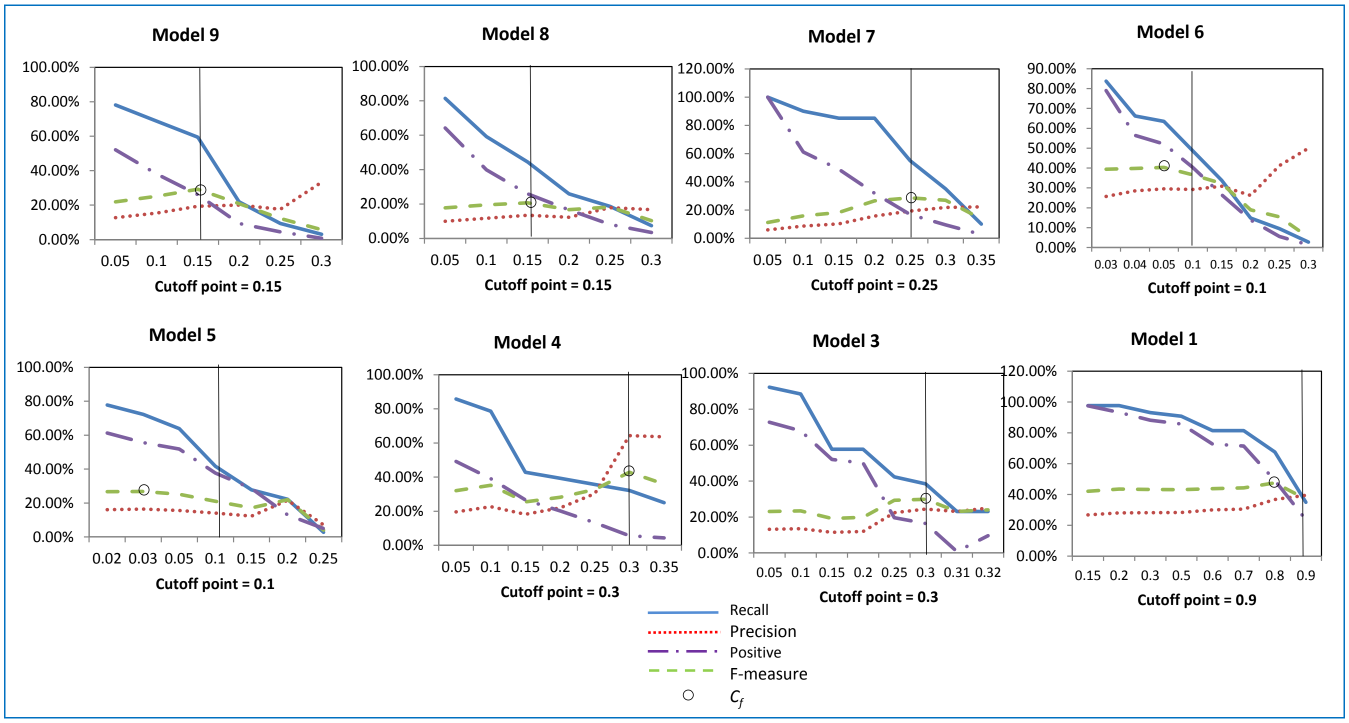

Figure 4. Recall, Precision, Positive and F-measure for different cutoff points

the highest *F-Measure* can also indicate a high ratio for the number of requirements predicted to be changed to the total number of requirements, which is categorized as *positive*.

For example, in Model 6, the highest cutoff point c_f is 0.05. Although the model can recall 63.51% of the requirements that are actually changed in the next version, the positive percentage is 52.13%, which means that the model predicts more than half of the requirements to be changed. In order to keep the *positive* percentage in an appropriate range, it is necessary to balance the *positive* and *F-Measure* as well.

Because predicting most of the requirements to be changed is meaningless, we define 50% as an upper limit for positive percentage to limit the proportion of the requirements predicted to be changed. Given the c_f value, we further look at the positive percentage of that point. If the *positive* percentage of that point is less than the upper limit, then c_f can be accepted as the final cutoff point. If the positive percentage of that point is larger than the upper limit, then we find the cutoff point that has the nearest positive value to the upper limit.

For example, in Model 6, the *positive* percentage for the highest cutoff point c_f is 52.13%. Then, we find the nearest *positive* percentage 40.33%, and we select the corresponding 0.1 as the final cutoff point.

After selecting the cutoff points for the 8 models, we evaluate their prediction performances in term of the *recall*, *precision*, *positive*, and *F-Measure*. In Figure 4, we can observe that all of the cutoff points, except for that in Model 1, are less than 0.5, and the cutoff point for Model 1 reaches 0.9. Model 1 is trained by a 3-version evolution dataset. The high cutoff point indicates that early prediction models with less historic information tend to output a high probability for future changes.

We can consider model 9, which has the highest recall value. As described in TABLE VI, it is trained by evolution data before version 11, and predicts volatile requirements of version 12. The *positive* percent is 26.06%, which means that 26.06% of the requirements in version 12 are predicted to change in version 13. The results can help save effort when reviewing the remaining 73.94% requirements to identify volatile requirements. The value of *precision* is 19.39%, which means that 19.39% of the predicted possible changing requirements are correct. There are 59.39% actual changes hit in the predicted results, which means that most of the actual changes can be found in advance.

In Figure 5, we can also observe that using the selected predictors can achieve an increasing performance on the *recall* values.

Model 4 has the lowest recall value among all of the models. Model 4 obtains only 32.14% of the actual changes. However, this model has the highest (64.29%) *precision* value and the lowest (5.65%) *positive* percentage, which means that it can

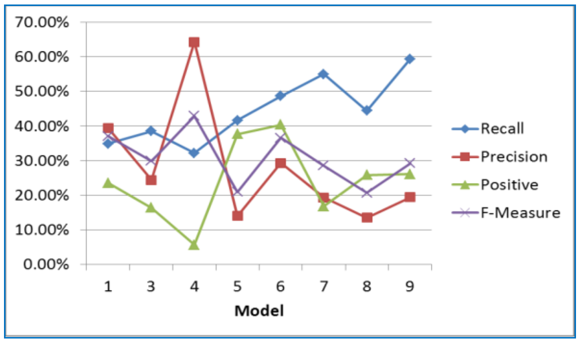

Figure 5 The prediction performances of the selected cutoff points

find 32.14% of the actual changes solely under the prediction of the 5.65% requirements to be changed. The low positive percent is also valuable because it can help to save effort on analysis of volatile requirements, especially when the size of the requirements is relatively large.

Model 6 has the highest *positive* percent 40.33%, which means that the model tends to predict more requirements to be changed in the future. The *recall* of the actual changes is 48.65%, which could relate to the high positive percent. However, the *precision* 29.27% is also relatively high, which means that almost one third of the predicted possible changes are correct.

TABLE IX shows the descriptive statistics for the *recall*, *precision*, *positive*, and *F-Measure*. There are 24.05% requirements that are predicted to change on average. The average *precision* for the predicted possible changing requirements is 27.95%. The prediction models can retrieve 44.33% of the actual changed requirements on average. We can observe that the deviation of the *F-Measure* is relatively small, which is the tradeoff result of the cutoff point selection.

In summary, prediction models built based on evolution history are capable of predicting future requirements changes. Based on our experiment results, *topicVolatility*, *frequency*, *lifecycle*, and *occurrence* are suitable predictors for predicting the product in this case study. The prediction performances are relatively efficient, at least under the case of this study.

TABLE IX DESCRIPTIVE STATISTICS FOR THE PREDICTION MODELS

	Max	Min	Mean	Std. Dev.
Recall	59.38%	32.14%	44.33%	0.095
Precision	64.29%	13.48%	27.95%	0.170
Positive	40.33%	5.65%	24.05%	0.114
F-Measure	42.86%	20.69%	30.72%	0.078

IV. DISCUSSION AND FUTURE WORK

In this section, we discuss issues that are related to the prediction method and future works of our study.

Balancing the requirements analysis costs and risks. Volatility analysis on the overall requirements is an important task in requirements evolution management. Instead of analyzing the overall requirements, our result can narrow down the scope to a small set of the requirements. Focusing on these requirements could still achieve a 44.33% *recall* value. The prediction results are particularly valuable when there are many requirements. It will be valuable if we can further improve the *recall* value. In addition, a cutoff point can be selected that balances *positive* and *recall*. Development teams such as agile teams that have tight resources on requirements analysis can require a cutoff point with a lower *positive* percentage, to control the efforts, but it could afford higher risks of leaving potential changes to the future. Development teams with sufficient resources can choose a cutoff point with a higher *positive* percentage to find more changes, and thus lower the potential risks of future evolution.

Weighted recall and precision. In our study, we use the *F-Measure* to consider the *recall* and *precision* to be equally important. However, the decision can be different according to different types of development teams. Development teams with sufficient resources can consider the *recall* to be more important than *precision*, which can better control the impact of requirement changes. Development teams that have tight resources can consider the *precision* to be more important. In such situations, a balanced F-Measure [30] can be applied.

Natural Language analysis on the requirements. The *topicVolatility* metrics can be easily calculated because the requirements used in this study are explicitly labeled with a corresponding topic. For a dataset without such a property, some machine learning techniques can help to cluster textual requirements into different topics [4].

Quantitative requirements management. Our study can benefit companies that are devoted to high-maturity process improvement. For example, companies that achieve CMMI [29] Level 4 or Level 5 could manage their requirements changes in a quantitative way. They can collect evolution history on their own products and establish prediction models that can predict future requirement changes based on the methodology described in Section II.

We recognize that one possible negative effect of our study is that some important requirements changes may be missed by the prediction results. In future work, we attempt to raise the *recall* by improving the *topicVolatility* metric. We plan to incorporate the priorities of requirements into the metric. We also consider incorporating evolution-related requirements categories such as difficult-to-learn functionalities into the metric.

V. THREATS TO VALIDITY

The **internal validity** of our study arises from the metrics we defined. In statistics, it is generally suggested that explanatory variables of logistic regressions should be independent with each other. In our study, we use a combination metric *occurrence* as one of the predictors. We observed that this metric did not have conflicting impacts with other metrics, and could get a better prediction outcome. Therefore, the threat to internal validity is limited.

One **external validity** issue lies in the fact that the case study is conducted on one single product. An obvious threat is that the prediction model trained in this study cannot directly generalize to other software products. The threat can be conveniently mitigated by retraining the prediction model on a given product to make useful predictions for the evolution of that product. However, for those new products with no historic evolution information, the prediction model is not applicable. The history-based prediction model is only valid on products in which the requirements go through multiple iterations prior to development. In this study, we apply our prediction approach to textual requirements. Another threat for generalization arises from that whether the prediction method can be applied on visual-related requirements, such as UML. The threat can be alleviate by taking advantage of visual transformation tools. For example, UML diagrams can be transformed into XML format by SPARX EA[4] and IBM Rational Rose[5]. Thus, practitioners can use the transformed XML files as inputs of the prediction approach in the same way as the textual requirements.

The **construct validity** of our work arises primarily from the requirements changes that we identify. We identify requirements changes by comparing requirement documents of two successive versions using tools that provide comparison, such as WinMerge[6]. Varying the requirements formats of two successive versions can influence the accuracy of the changes that we identified. For example, development teams can use textual requirements that were written in natural language in earlier versions but that can be turned into UML graphs in later versions. It is even difficult to trace the link between two identical requirements, as well as the changes. In that case, we collect the change logs for each version and manually identify the changes from the change logs. To assure the correctness of the requirement changes, we ask practitioners in the case company to confirm and correct our identifications. Another threat arises that when a requirements specification evolves, it is likely that some changes are solely changes in the representation, without changing the content. In this study, we alleviate the threat by manually filtering out the trivial changes, such as typos. The third threat to the construct validity is regarding to the *topicVolatility* metric that we measured. In the case company, the tasks of grouping requirements by topics are performed by the requirements analysts during the phase of release planning. The results may contain deviations because it involves human judgments. Because most of the requirements analysts are experienced staffs and domain experts, we consider the deviations on the measurement as a minor threat.

The **conclusion validity** of our study arises from the two hypotheses we made. We assume that frequently changed requirements are likely to change in the future. In the case company, the drawn conclusions are consistent with the hypotheses. There is a risk that the hypotheses do not hold in other companies, since the training data may show different evolution patterns in different industrial settings.

VI. Related Work

Our study is related to previous research as follows:

Requirements evolution prediction. Bush and Finkelstein [6][7] proposed a qualitative analytical approach to assessing the requirements stability by modeling possible future worlds in a delicate analysis with a goal-based model with scenario descriptions of requirements. They addressed the issue of the analysis of individual requirements changes, which is especially useful for important changes. In this study, we facilitate the change analysis on the overall requirements, which is especially useful when the requirements sets are large. Loconsole and Borstler [20][21] collected requirements from a medium-sized software project and used the size of the requirements as predictors of the requirements volatility. They predicted the total number of changes in the requirements, while we predict exactly which requirements might change in the future. Yang et.al identified the requirements that have intrinsic uncertainty and ambiguity by using natural language and machine learning techniques [32][33]. They identify volatile requirements from natural language descriptions, while we statistically identify volatile requirements from history information. Comparing to their work, our study addresses the prediction problem from a different viewpoint.

Requirements evolution measurement. Researchers have provided a series of evolution metrics to analyze evolution characteristics. Stark et al. [28] presented the source, frequency and types of changes in the project plan process of a large software system. Anderson and Felici [1][2][3] conducted several empirical investigations on requirements evolution of an avionics industrial safety-critical case study with metrics, identifying evolutionary product features and other interesting aspects of changing requirements. Nurmuliani et al. [25][26] conducted a case study that characterizes the nature of requirements volatility with a classification of requirements changes and visualized the extent of the requirements volatility during the software development lifecycle. Gall et al. [12] tracked the historic evolution of several releases of a telecommunications switching system and discovered the difference in the behavior of the whole system versus its subsystems. Wang et al. [31], Costello and Liu [10] quantitatively measured the requirements volatility in terms of the number of changes (additions, deletions and modifications). Henry and Henry [16] used a number of specification changes, such as the average changed SLOC and average changed modules, to measure the requirements volatility. Henderson-Sellers et al. [15] selected the size of use cases, environment factors, and measurement of atomic actions as measures of requirements volatility. Javed et al. [17] measured requirements volatility by pre/post-functional specification changes and post -release changes. Lam and Shankararaman [18] used the change effort, change volatility, change completeness, change error rates, and change density as requirements volatility metrics. Our metrics focus on characterizing the history of the requirements evolution from three aspects: the topic to which the changed requirements relate, the degree of the change, and the length of the change time, which complements the prior study.

VII. Conclusions

In this paper, we propose a methodology for predicting requirements evolution based on historic information. Based on this methodology, we conducted a case study on the requirements evolution of an eight-year product from an industrial company. We built a prediction model by using the following predictors: *topicVolatility*, *frequency*, *lifecycle*, and *occurrence*. After balancing between the analysis workload and evolution risks, we provide the prediction results as a set of possible changing requirements in the future. There are 24.05% requirements that are predicted to be changed on average. The average precision for the predicted possible changing requirements is 27.95%, and we can retrieve 44.33% actual

[4] http://www.sparxsystems.com/products/ea/index.html

[5] http://www-01.ibm.com/software/rational/

[6] http://winmerge.org/

changed requirements on average. In summary, the results show that prediction models that are built based on evolution history are capable of predicting future requirement changes, and allow practitioners to better control the requirements evolution.

Acknowledgment

We appreciate anonymous reviewers for their supportive and constructive comments. We would also like to thank Ye Yang, Yan Sun, Junji Zhi, and Hong Wu with the writing. We acknowledge the support of National Natural Science Foundation of China No. 61073044, 60803028, and National Science and Technology Major Project of China No. 2010ZX01036-001-002, and 2010ZX01037-001-002.

References

[1] Anderson, S. and Felici, M., "Requirements evolution from process to product oriented management," in Proc. 3rd PROFES, pp. 27-41, 2001.

[2] Anderson, S. and Felici, M., "Controlling requirements evolution: An avionics case study," in Proc. 19th SAFECOMP, pp. 361-370, 2000.

[3] Anderson, S. and Felici, M. "Quantitative aspects of requirements evolution," in Proc. 26th COMPSAC, pp. 27-32, 2002.

[4] Ayad, H. and Kamel, M., "Topic Discovery from Text Using Aggregation of Different Clustering Methods," in Proc. of 15th AI, pp.161-175, 2002.

[5] Boehm, B.W. "Improving software productivity", IEEE Comput. 20(9), pp. 43-57, 1987.

[6] Bush, D. and Finkelstein, A., "Environmental Scenarios and Requirements Stability", in Proc. IWPSE-EVOL, pp. 133-137, 2002.

[7] Bush, D. and Finkelstein, A., "Requirements Stability Assessment Using Scenarios", in Proc. 11th ICRE, pp. 23-32. 2003.

[8] Chambers, J.M. and Hastie, T.J., "Statistical Models in S", Wadsworth & Brooks, Pacific Grove, Calif., 1992.

[9] Chen, K.; Zhang, W.; Zhao, H. and Mei, H., "An approach to constructing feature models based on requirements clustering", in Proc. 13th ICRE, pp.31-40, 2005.

[10] Costello, R.J. and Liu, D. "Metrics for Requirements Engineering," Journal of Systems and Software, 29(1), pp. 39-63, 1995.

[11] Davis, A.M.; Nurmuliani, N.; Sooyong Park; Zowghi, D., "Requirements Change: What's the Alternative?", in Proc. 32nd COMPSAC, pp. 635-638, 2008.

[12] Gall, H.; Jazayeri, M.; Klosch, R. and Trausmuth, G., "Software evolution observations based on product release history," in Proc. 13th ICSM, pp. 160-166, 1997.

[13] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; "Design Patterns: Elements of Reusable Object-Oriented Software, Reading", Mass.: Addison-Wesley, 1995.

[14] Harker, S D P.; Eason, K D. and Dobson, J E., "The change and evolution of requirements as a challenge to the practice of software engineering", in Proc. 1st ICRE, pp. 266-272, 1993.

[15] Henderson-Sellers, B.; Zowghi, D.; Klemola, T. and Parasuram, S., "Sizing Use Cases: How to Create a Standard Metrical Approach", in Proc. 8th OOIS, pp. 409-421, 2002.

[16] Henry, J. and Henry, S., "Quantitative Assessment of the Software Maintenance Process and Requirements Volatility", in Proc. ACM CSC, pp. 346-351, 1993.

[17] Javed, T.; Maqsood, M. and Durrani, Q.S., "A Study to Investigate the Impact of Requirements Instability on Software Defects", ACM SIGSOFT Software Engineering Notes, 29(3), pp. 1-7, 2004.

[18] Lam, W. and Shankararaman, V., "Managing Change in Software Development Using a Process Improvement Approach", in Proc. 24th EUROMICRO, pp. 779-786, 1998.

[19] Lim, S.L. and Finkelstein, A., "Anticipating Change in Requirements Engineering.", Journal of Relating Software Requirements and Architectures, pp. 17-34, 2011.

[20] Loconsole, A. Borstler, J., "An Industrial Case Study on Requirements Volatility Measures," in Proc. 12th APSEC, pp.249-256, 2005.

[21] Loconsole, A. Borstler, J., "Construction and Validation of Prediction Models for Number of Changes to Requirements". Technical Report, UMINF 07.03, Feb. 2007.

[22] Long, J S., "Regression models for categorical and limited dependent variables". SAGE Publications, Incorporated, 1997.

[23] Malaiya, Y.K. and Denton, J., "Requirements Volatility and Defect Density", in Proc. 10th ISSRE, pp. 285-294, 1999.

[24] McGee, S. and Greer, D., "Towards an understanding of the causes and effects of software requirements change: two case studies." Requir. Eng. 17(2), pp. 133-155,2012.

[25] Nurmuliani, N.; Zowghi, D. and Fowell, S., "Analysis of Requirements Volatility during Software Development Life Cycle", in Proc. ASWEC, pp.28-37, 2004.

[26] Nurmuliani, N.; Zowghi, D. and Williams, S. P., "Characterising Requirements Volatility: An Empirical Case Study", in Proc. ISESE, pp. 427-436, 2005.

[27] Ruhe G, Saliu M O. "The Art and Science of Software Release Planning", Software, IEEE, 22(6): 47-53, 2005.

[28] Stark, G. E.; Oman, P.; Skillicorn, A. and Ameele, A., "An examination of the effects of requirements changes on software maintenance releases," in Journal. of Software Maintenance: Research and Practice, 11(5), pp. 293-309, 1999.

[29] SEI, Capability Maturity Model Integration (CMMI), Version 1.3. Staged Representation. 2010, Carnegie Mellon University, Software Engineering Institute: Pittsburgh.

[30] van Rijsbergen, C. J., "Information Retrieval (2nd ed.)". Butterworth, 1979.

[31] Wang, H.; Li, J.; Wang, Q. and Yang, Y. "Quantitative analysis of requirements evolution across multiple versions of an industrial software product," in Proc. 17th APSEC, pp. 43-49, 2010.

[32] Yang, H.; De Roeck, A.; Gervasi, V.; Willis, A.; Nuseibeh, B. "Analysing anaphoric ambiguity in natural language requirements". Requirements Engineering, 16(3), pp.163-189, 2011

[33] Yang, H.; De Roeck, A.; Gervasi, V.; Willis, A.; Nuseibeh, B. "Speculative requirements: Automatic detection of uncertainty in natural language requirements" in Proc. 20th ICRE, pp. 11-20, 2012.