Prediction of the Successful Completion of Requirements in Software Development—An Initial Study

Witold Pedrycz, Joana Iljazi, Alberto Sillitti and Giancarlo Succi

Abstract A lot of requirements are discarded throughout the product development process. However, resources are invested on them regardless of their fate. If it would exist a model that predicts reliably and early enough whether a requirement will be deployed or not, the overall process would be more cost-effective and the software system itself more qualitative, since effort would be channeled efficiently. In this work we try to build such a predictive model through modelling the lifecycle of each requirement based on its history, and capturing the underlying dynamics of its evolution. We employ a simple classification model, using logistic regression algorithm, with features coming from an engineering understanding of the problem and patterns observed on the data. We verify the model on more than 80,000 logs for a development process of over 10 years in an Italian Aeronautical Company. The results are encouraging, so we plan to extend our study on one side collecting more experimental data and, on the other, employing more refined modeling techniques, like those coming from data mining and fuzzy logic.

1 Introduction

During the typically long lifetime of a software system, certain requirements are frequently modified or even completely deleted, due to factors like change of business needs, increase of the complexity of the system itself, changes in mutual dependencies, or disability to deliver the service [1, 2]. However, financial resources, human and machine effort and time is invested on these requirements,

W. Pedrycz · J. Iljazi
Department of Electrical and Computer Engineering,
The University of Alberta, Edmonton, Canada

A. Sillitti
Center for Applied Software Engineering, Genoa, Italy

G. Succi (✉)
Innopolis University, Innopolis, Russia
e-mail: G.Succi@innopolis.ru

G. Jezic et al. (eds.), *Agent and Multi-Agent Systems: Technology and Applications*, Smart Innovation, Systems and Technologies 58,
DOI 10.1007/978-3-319-39883-9_21

regardless their fate and final state. If there would be a model that could predict early enough on the development life cycle that some requirements are more prone to failure than others, the management of the processes would be more cost-effective and deterministic [3]. This paper is focused on trying to predict such failure using a suitable and simple mathematical model.

Empirical software engineering is an established discipline that employs when needed predictive modeling to make sense of current data and to capture future trends; examples of use are predictions of system failures, requests for services [3], estimation of cost and effort to deliver [4, 5]. With respect to requirement modeling, empirical software engineering has so far put most effort in lifecycle modeling [6–8] and volatility prediction [9, 10]. In our view, the prediction of the fate of requirements and of the associated costs has not received an adequate attention; in this work we propose this novel viewpoint that centers its attention on this neglected area of study.

In this work we use logistic regression to learn from historical sequences of data, each sequence with a final label of success or a failure depending on its final evolution state. in an industrial software system development process. The model is experimented on a real data coming from 10 years of requirements properly evaluated in an Italian Aeronautical Company. The experimental data features more than 6,000 requirements and 80,000 logs of their evolution.

As it is shown further, this results in a statistically significant model that is able to detect failures and successes with a satisfactory precision. This analysis opens the path for a future better understanding of the requirement lifecycle dynamics, the identification of the typical patterns that show a future failure, and a broader exploration of classifying techniques for data of software engineering processes. Moreover, keeping in mind the limitations of the model we build, the next step would be also a consideration of the trade-off between specificity and out of sample generalization.

The paper is structured as follows: in Sect. 2 we present the state of the art, in Sect. 3 the methodology followed in this early study is detailed, highlighting the prediction method chosen and the way its performance is assessed. In Sect. 4 it is presented the case study. The paper continues further with Sect. 5 where the limitation of this work are outlined and the future work is presented. The paper is concluded (Sect. 5) by summarizing the results of this work and its importance.

2 State of Art on Empirical Studies on Requirements

Requirements are an important part of the software development process; some of their characteristics are studied broadly and the results of such studies are widely present in the literature. However, as mentioned in the introduction, there is more to explore especially in terms of empirical studies of requirements evolution [11]: in this context the existing research is focused on two main directions. The first is modeling the requirements evolution with the aim of understanding its dynamics

early on their specification [6] or during the lifecycle [8, 12], considering important also developing tools to capture this evolution through logs of different stages of development process [13].

The second direction has for long focused on the requirements volatility. There are studies identifying some main reasons that can take a requirement not to succeed or to change frequently, business needs, requirement complexity and dependency, environment of development, etc. [1, 2]. To the volatility issue, a few studies approach it, as a reflection of the socio-technical changes in the world and in industry. For instance, Felici considers changes not an error to deal with, but an unavoidable part of software process development that reinforces the requirement itself and saves the project from failing, by adapting continuously. In this case except the cost accompanying the requirements volatility, the authors try to associate also the benefit driven from the changes [14–16]. There is also a part of studies that give importance to how changes propagate [17], and how they can be related to an increase in defects present and in the overall project cost [18].

With respect to learning from historical data, there are empirical software engineering studies attempting to build predictive models on requirements, some using simple regression models for identifying which are the future volatile trends and which requirements are more prone to change [9, 10], other using different alternatives like building a rule-based model for prediction [7]. Describing changes and stability points through scenario analysis is also another proposed solution [19].

In any case, what is relevant to stress with reference to our work, is that often the deleted requirements are not taken in consideration at all in these analysis, or the deletion process itself is considered simply as one of the changes that may happen [10, 14]. Thus, despite being a fairly easy concept to grasp and challenging to work with, there is almost no literature referring to failing requirements or trying to model the requirement final state. We think that our unicity gives more value to this paper since it shift the attention toward a field explored so little.

3 Methodology

The main goal of this study is to answer to the research question: “Can we predict reliably the final status of a requirement of a software system analysing its historical evolution through stages and the development environment? If so, how early on the life cycle of the requirement can we perform the prediction?” Our aim is to build a model to make this possible. We start by making a relevant assumption, that requirements are independent of each other, meaning that the status of one requirement does not affect the status of other requirements. This assumption is quite stringent, and for certain perspective also unfeasible in the real world, still it is the basis of the followup analysis, and, as we will see, its violation does not affect the validity of our findings.

3.1 Prediction Method

Machine learning is usually the right technique when we do not have an analytical solution and we want to build an empirical one, based on the data and the observable patterns we have [20]. Inside this field, supervised learning is used when this data is labeled, meaning the dependent variable is categorical. This is also our case, since we are dealing with a dichotomy problem, a requirement being successful or failed, thus the main choice is to use supervised learning algorithms in order to build a predictive model. Applying logistic regression algorithm, from the supervised learning pool, would give not only classification of the requirement objects into pass/fail, where pass stands for a success and fail for a never deployed requirement, but also the probability associated with each class.

Logistic regression is an easy, elegant and yet broadly used and known to achieve good results in binary classification problems. Assuming that we have d predictors and n observations, with X being the matrix of observations and Θ the matrix of the weights:

$$\text{for} \quad X = \begin{bmatrix} x_{11} & \dots & x_{1d} \\ \dots & & \\ x_{n1} & & \end{bmatrix} \quad \theta = \begin{bmatrix} 1 \\ d \end{bmatrix}$$

the LR model is $logit(X\theta) = \frac{1}{1+e^{-X\theta}}$.

In order to learn Θ, the algorithm minimises the uncertainty, by first assuming an independence between the observations. The result of the iterations is the discriminant that divides the input space into pass and fail area. Thus, per any new requirement data point, once learned the model and tuned with the right threshold, we would be able to give a statistically significant probability value of it being a future pass or fail.

3.2 Evaluation of Performance of Predicting Models

To assess the performance of the classifier we use the confusion matrix. The confusion matrix values allow us to calculate the predicting ability and error made in terms of type I and type II error. Clearly, since logistic regression delivers a probability of belonging to a certain class, as we discussed in the section above, in order to determine the label a threshold is used. As the threshold changes, also the ability of classifiers to capture correctly passess and failures is channeled (inclined/biased). We use the ROC curve analysis, through maximising the combination of sensitivity-specificity of our model we obtain the best threshold value and tune the model accordingly. Thus, the metrics we will use to measure performance are: sensitivity, specificity and the Area Under the Curve. Sensitivity shows the ability of the model to capture correctly fail-requirements in the dataset,

whereas the Specificity shows the ability to capture passes. The area under the curve is calculated by plotting Sensitivity versus Specificity for threshold value from 0 to 1, whereas the formulas to calculate these two are below:

Confusion matrix	Predict fail	Predict pass
Real fail	True Positive	False Negative
Real pass	False Positive	True Negative

$$Sensitivity = \frac{TruePositive}{TruePositive + FalseNegative}$$
$$Specificity = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

4 Case Study

4.1 Data Description and Processing

The data we use in this research come from an industrial software system, developed in an aeronautical company in Italy, Alenia Aermacchi. The data of the development process are formated as a file of logs, presenting the history of 6608 requirements and 84891 activities performed on them on a time period from March 2002 to September 2011. Every log is characterized by the following attributes summarized in the table:

Timestamp	Date and time in which the requirement was firstly created in the log system
Version	Version of the software product in which the requirement was first created
Developer	Developer that performed this specific action on the requirement object, and entered it in the log system
Activity	The specific activity performed on the requirement
Requirement ID	The identification number, unique per each requirement

There are 13 different activities performed on a requirement throughout the development process, with the most significant numerically and interesting from the process point of view being:

createObject	The requirement is created in the system
createLink	Creates a link between different requirement objects and keeps count of it
modifyObject	The requirement is modified
deleteObject	The requirement object is deleted

The early identification of "deleteObject" is basically what we are after in this study, since it means that the requirement was deleted from the system without ever being deployed, this is what we will call a "fail". Every other activity concluding the lifecycle of the requirement present in database is considered a success, and the requirement itself a "pass". At this point is clear that once processed, we get binomial labeling for the attributes characterizing the requirements lifecycle inside this process. From a first statistical analysis, we notice that there is a dominance of "modifyObject" activities in the requirements labeled as "fail." The early attribute selection is performed based on domain expertise and influenced by any possible pattern that we identified, as mentioned above. The processed final requirement records are the rows of a data frame whose columns are the potential features of the predictive model we aim to build, except the ID that is used to keep track of the specific object. Each requirement record is the characterized by elements presented in the following table.

Requirement ID	The identification number
Opening version	The version where it was first created; it will also ensure a timeline on the requirement objects
Developer	The last developer that worked with the requirement
Number of transitions	The cumulative number of activities performed on the requirement; this is also an approximation of the number of modification a requirement goes through.
Lifetime duration	The longevity of a requirement, calculated as the difference between the closing and the opening dates
Final state	The label either pass or fail

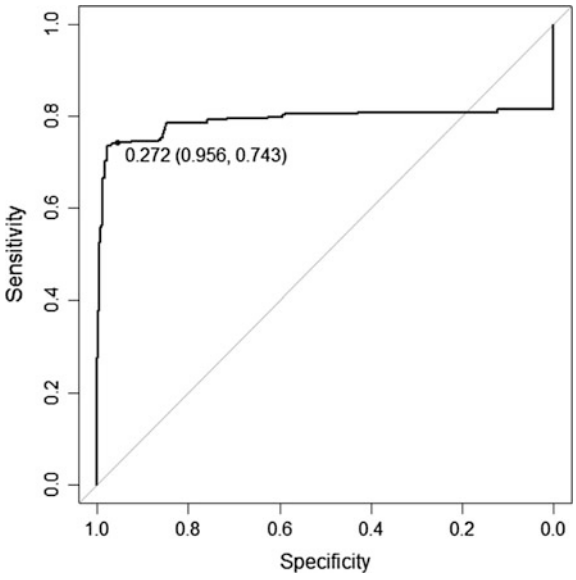
4.2 Learning of the Model and Evaluation of Performance

We start with a slightly imbalanced time-ordered dataset of requirements. Each record of it is a requirement object profiled by four features that we will try to use in our model: version, number of Transitions, lifetime, developer. We allow the lack of balance in order to give an initial bias toward the pass-class, considering that, however, a false-pass costs less to the system than a false-fail. To learn the model, stepwise regression is performed on the data. The statistically significant features

used by the model are: version, number of transitions and lifetime of the requirement. The steps followed are:

1. Dataset is divided into training and validation sets(respectively 75 %–25 %), preserving the initial class distribution present on the original data, using package "caret"
2. Logistic regression algorithm is run (10-k cross validation, repeated 3 times) on training data to build the model.
3. ROC curve is plotted in order to find the best sensitivity-specificity values by using the criteria "most top-left" on the roc curve. We notice: 79 % predictive ability of the model (AUC = 0.79 with 95 % CI) and the best threshold being 0.27
4. Model is tested on the validation set, where for the best threshold value, the results achieved in terms of predictivity are: sensitivity = 0.74 and specificity = 0.95

Thus, the model built, out of all true failures can capture correctly 74 % of them and out of passes 95 % of them, with the results being under 95 % CI. Below the plot:



This classification model built and tested on the whole dataset of requirements shows clearly for a possibility of predicting pass and fail requirements.

5 Limitations and Future Work

We think that this work presents a novel idea that can move forward in several directions, especially taking into account its limitations. The first limitation is inherent with the single case study. This work has a very significant value given the size of its dataset—a very rare situation in Software Engineering, and a quite unique situation in the case of requirements evolution, however, it is a single case, and, as usual a replication would be strongly advisable. The second limitation is that we have assumed the requirements independent, while they are not. This situation is similar to the one in which a parametric correlation is used while clearly its requirements are not satisfied—still it has been experimentally verified that the predictive value still hold, as it holds in this specific case. The third limitation is that we build a model having available the whole stories of the requirements, while it would be advisable to be able to predict early whether a requirement is going to fail. Therefore, the future studies should focus on trying to build predictive models using only initial parts of their story. The fourth limitation is that we do not take into account that the information on requirements is collected manually by people, so it is subject to errors, and this study does not take into consideration how robust the requirements are with respect to human errors. So a follow-up study should consider whether a presence of an error, for instance in the form of a white noise altering the logs of the requirements, would alter significantly the outcome of the model. A future area of study is then to apply more refined techniques from Machine Learning and from Fuzzy Logic to build a predictive system. The joint application of Machine Learning and Fuzzy Logic would resolve by itself the last three limitations mentioned above, also building a more robust and extensible system.

6 Conclusions

In this work we have presented how we can analyse the evolution of requirements to predict those that will fail, thus building a model that could to decide where to drop the effort of developers. The model is based on logistic regression and has validated on data coming from an Italian aeronautical company, including more than 6,000 requirements spanning more than 10 years of development. The results of this early study are satisfactory and pave the way for followup work, especially in the direction to make the model predictive, more robust, employing more refined analysis techniques, like those coming from Data Mining and Fuzzy Logic.

Acknowledgments The research presented in this paper has been partially funded by Innopolis University and by the ARTEMIS project EMC2 (621429).

References

1. Harker, S.D.P., Eason, K.D., Dobson, J.E.: The change and evolution of requirements as a challenge to the practice of software engineering. In: Proceedings of 1st ICRE, pp. 266–272 (1993)
2. McGee, S., Greer, D.: Towards an understanding of the causes and effects of software requirements change: two case studies. Requir. Eng. **17**(2), 133–155 (2012)
3. Succi, G., Pedrycz, W., Stefanovic, M., Russo, B.: An investigation on the occurrence of service requests in commercial software applications. Empirical Softw. Eng. **8**(2), 197–215 (2003)
4. Boehm, B., Abts, C., Chulani, S.: Software development cost estimation approaches—a survey. Ann. Softw. Eng. **10**, 177–205 (2000)
5. Pendharkar, P., Subramanian, G., Rodger, J.: A probabilistic model for predicting software development effort. IEEE Trans. Softw. Eng. **31**(7) 2005
6. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of 3rd IEEE International Symposium on Requirements Engineering, pp. 226–235, 6–10 Jan 1997
7. Le Minh, S., Massacci, F.: Dealing with known unknowns: towards a game-theoretic foundation for software requirement evolution. In: Proceedings of 23rd International Conference: Advanced Information Systems Engineering (CAiSE 2011). London, UK, 20–24 June 2011
8. Nurmuliani, N., Zowghi, D., Fowell, S.: Analysis of requirements volatility during software development lifecycle. In: Proceedings of ASWEC, pp. 28–37 (2004)
9. Loconsole, A., Borstler, J.: Construction and Validation of Prediction Models for Number of Changes to Requirements, Technical Report, UMINF 07.03, Feb. 2007
10. Shi, L., Wang, Q., Li, M.: Learning from evolution history to predict future requirement changes. In: Proceedings of RE 2013, pp. 135–144
11. Ernst, N., Mylopoulos, J., Wang,Y.: Requirements Evolution and What (Research) to Do about It. Lecture Notes in Business Information Processing, vol. 14, pp. 186–214 (2009)
12. Russo, A., Rodrigues, O., d'Avila Garcez, A.: Reasoning about Requirements Evolution using Clustered Belief Revision. Lecture Notes in Computer Science, vol. 3171, pp. 41–51 (2004)
13. Saito, S.; Iimura, Y., Takahashi, K., Massey, A., Anton, A.: Tracking requirements evolution by using issue tickets: a case study of a document management and approval system. In: Proceedings of 36th International Conference on Software Engineering, pp. 245–254 (2014)
14. Anderson, S.; Felici, M.: Controlling requirements evolution: an avionics case study. In: Proceedings of 19th SAFECOMP, pp. 361–370 (2000)
15. Anderson, S.; Felici, M.: Requirements evolution from process to product oriented management. In: Proceedings of 3rd PROFES, pp. 27–41 (2001)
16. Anderson, S.; Felici, M.: Quantitative aspects of requirements evolution. In: 26th Annual International Computer Software and Application Conference, COMPSAC 2002, pp. 27–32, 26–29 Aug 2002
17. Clarkson, J., Simons, C., Eckert, C.: Predicting change propagation in complex design. In: Proceedings of DETC'01, ASME 2001 Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Pittsburgh, Pennsylvania, 9–12 Sept 2001
18. Javed, T., Maqsood, M., Durrani, Q.S.: A study to investigate the impact of requirements instability on software defects. ACM SIGSOFT Softw. Eng. Notes **29**(3), 1–7 (2004)
19. Bush, D., Finkelstein, A.: Requirements stability assessment using scenarios. In: Proceedings of 11th ICRE, pp. 23–32 (2003)
20. Yaser, S.A.M, Malik, M.I., Hsuan-Tien, L.: Learning from data. http://www.amlbook.com/support.html