Early Outcome Prediction of Software Projects using Software Defects and Machine Learning

Michael T. Shrove
Millennium Corporation
Huntsville, Al.
tshrove@gmail.com
0000-0002-6907-5807

Dr. Emil Jovanov
University of Alabama Huntsville, ECE Dept.
Huntsville, Al.
emil.jovanov@uah.edu
0000-0001-6754-3518

Abstract: The goal of this research is to help software stakeholders predict early in the software project when or if the project is at risk of failure. If the decision makers can get an early notification of the outcome, they can make better choices on what they need to do to make the project successful. In this paper, we explore using the trend of defect totals as a function of the relative completion of the project. We collected the data from a software company who had multiple software projects that had a defined and consistent process and metric collection methods. We show how we used the defects totals from this data as features to a Support Vector Machine (SVM) to classify the project as successful or unsuccessful early in the project's lifecycle. We present our technique and methodologies for developing the inputs for the proposed model and the results of testing. Further, we discuss the prediction model and the analysis of an SVM model. We then evaluate the labels from the company's dataset to our prediction model and show that it demonstrates a prediction accuracy of 88.7% in a set of 13 projects.

Keywords: *Machine Learning, Software Quality Management, SVM, Defects, Project Outcome Prediction*

1. Introduction

In today's world, software has almost become a part of every job function in the market. From the food industry to the medical industry to the defense industry, software is everywhere. Which in turn, means more and more people rely on software that is of high quality and produces accurate results. However, in 2015 a report by Standish Group showed that only 29% of software projects are successful and 52% were "challenged" (Wojewoda et al., 2015). Even in the 52% that were "challenged", do we know they produced a quality product? A successful project does not necessarily mean a quality product. In (Lehtinen et al., 2014) the authors lay out four main high-level areas for why software projects fail: *People, Tasks*, *Environment*, and *Methods*.

In this study, one of the main areas that we focused on was the lack of software testing, which was directly related to software project failure. In turn, the lack of software testing can lead to more software defects which inevitably leads to software failure. Moreover, in (Guillaume-Joseph, 2015), the authors also found that a lack of software testing led a high rate of defects found in the production system and low-quality product.

Most of the existing research is focused on predicting defects in codebases of software projects, rather than using defects to assess trends of defects that can determine success or failure of the overall software project. In our research, we focused on trying to resolve the problem of software failure by providing the stakeholders with an indicator of the state of the project as early in the development cycle as possible. We use the patterns indicated by available resources (i.e. defects) to determine the likelihood of the project's failure. In section 2, we explain the goal of this research in relation to other publications in this field. In section 3, we describe our proposed approach,

algorithms, libraries, and methods used. Section 4 presents the result and analysis of the implemented method. Lastly, in section 5, we talk about factors that threaten the validity of the project.

2. Survey of Research in Software Engineering Outcome Prediction

With the increase in data being generated, so has the need and popularity in machine learning and statistical-based models. These models are increasingly being used in different market sectors. For example, business and stock markets are using these models to forecast sales and stock prices. Though, in our research, we want to classify the project as a success or failure by the total number of defects of a particular software project. In our literature review, we found that a majority of authors focusing in this area are trying to classify if a certain code module is or will be defect prone in the future based on software metrics (Fenton et al., 2007; Lessmann et al., 2008; Okutan, 2014; Song et al., 2011; Vashisht et al., 2015; Wang, 2013; Menzies et al. 2018). Time series trend forecasting seemed to be more in financial markets (BouHamad, 2019). But two papers, (Weber, 2003; Ramaswamy et al.) was very similar to our research. However, their approaches were different. In (Weber, 2003), they used a k-means clustering approach on projects within companies by stages of the projects while still using defects as the primary means of clustering. Meanwhile, (Weber 2003), used questionnaire data answered from software developers along with various prediction methods (i.e. k-NN) to predict the outcome of projects.

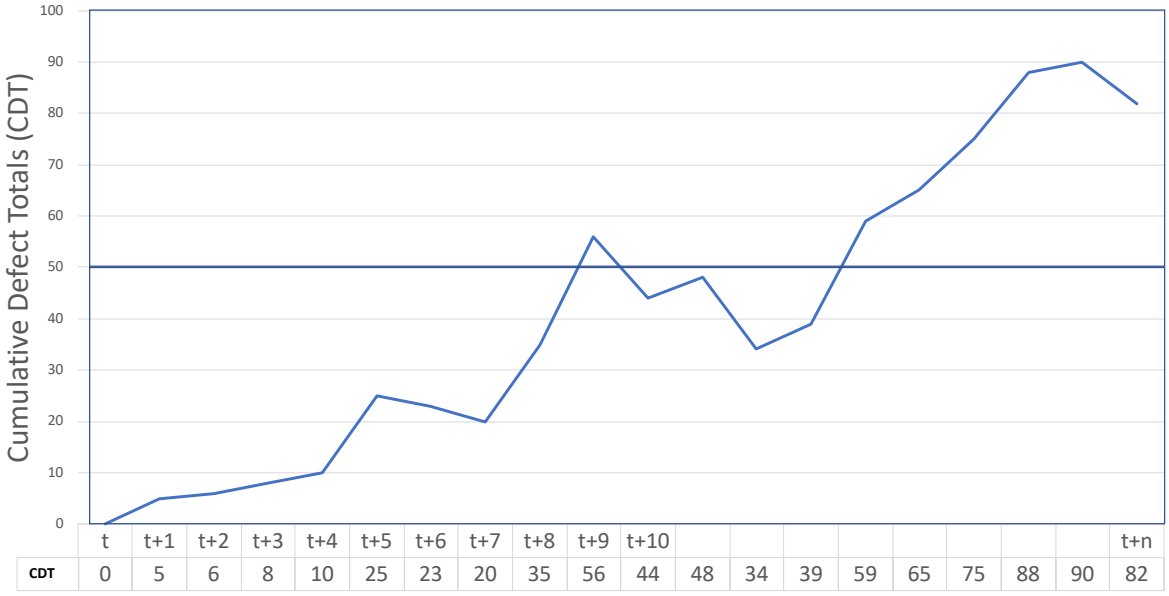
3. Outcome Prediction Model

In this section, we present an overview of the proposed model and the data set that was used to test the model. Techniques we used for identification and measuring defects from the training dataset are presented in section 3.3. Lastly, we will describe in detail the prediction technique and the algorithms related and used for predicting the outcome of a software project using the training dataset, in section 3.4.

3.1. Overview of Approach

The symbolic illustration of TD evolution during the course of a project is presented in Figure 1.

We will present a realistic evaluation of the defect dataset in section 3.2. Figure 1 represents the total defect count accumulated over time at time n from the start of the project, represented over the life of the software project. Each point can represent total defect count at that time in the



	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9	t+10								t+n
CDT	0	5	6	8	10	25	23	20	35	56	44	48	34	39	59	65	75	88	90	82

Fig.1- Illustration of our prediction model using evolving defects of a project over time

project either by sprints, releases, or custom times in the project that may be important for the decision makers. The horizontal line separating likely successful projects (bottom half/green) and likely failing projects (top half/red) is generated by the machine learning model based on past outcomes in software projects used in the training model. When the defect count reaches the threshold, the monitoring software would notify the project decision makers that the project is at risk of a failure and provide the confidence level of the decision. Project managements can make the decision to resolve the defects or alter the project to change the outcome. Together the decision maker, can decide to look deeper into the model to see what is making the model predict this outcome, and decide to change release plans or sprints plans to achieve a successful outcome in the project.

The overall approach of how we performed this implementation can be seen in Figure 2. We will describe in detail each stage of our approach in sections 3.2 – 3.4 and in section 4.

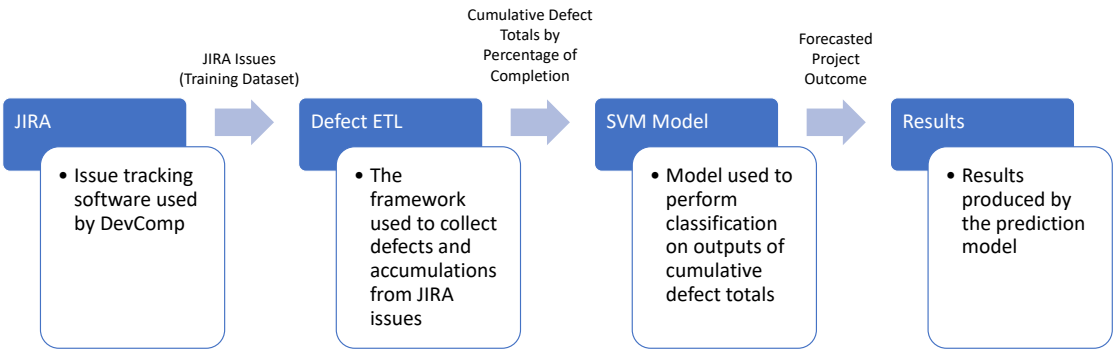


Fig.2 - High level view of workflow of data to model output

3.2. Training Dataset

The training dataset used for this research was one from a software-based company who we will call DevComp. DevComp has been tracking a subset of their software projects over the past 4.5 years using Atlassian JIRA as their issue tracking system. Their projects range from big data software to business web application software. Most projects in the dataset had a lifespan of 1 year. They have a very robust agile-based process and have a wide variety of issue types in JIRA that would keep track of different work items with more granular issue types instead of generic types. This granularity allowed us to better distinguish issues. The projects range from ongoing projects to projects that have finished and completed with the outcome defined by their internal process. There was a total of 13 projects. Out of the 13 projects, one was on-going (ProjM) and the remaining were completed. Three of the 13 were projects that had failed the others were successful. Four projects lasted more than 2 years, and the remaining projects were 3-20 months long (mean: 234 days ± 198 days). All 13 projects had a team of developers, designers, testers, and management, with 4 developers on average not including the other role types mentioned. Total number of personnel on each team on average was 7, including all roles. DevComp allowed us access to their Atlassian JIRA API to collect the data we needed in order to perform the analysis. We used an open source library jira-python (PyContribs, 2017) to extract datasets from DevComp's API.

3.3. Data Preprocessing

We used an automated estimate technique (Fernández-Sánchez et al., 2015) in our experiment to measure the defects. We wanted an automated mechanism that would identify, measure, and produce the results for the stakeholders without human interaction. To make the process as autonomous as possible, we needed an automated technique for measuring the defect cost. Looking through the training dataset given to us by DevComp, we decided to calculate the defect count as +1 (production of defect), -1 (reduction of a defect or resolving the defect), or 0 (no change in defects).

We arranged the defects by date from the earliest dated defect first, to the most recent dated defect last. We then created a cumulative sum of the defect by project. Our output from this stage, the example seen by Figure 1, would be a cumulative sum list of defects of each project sorted from the oldest to the youngest defect. This would be the input into our model.

After creating a list of ordered defects (positive value for new defects or negative value for resolution of a defect), a graph of the defects was created. Defect events can be grouped by releases, specific dates in the projects, or percentage completed in the project (e.g. 10%, 30%). In our research we chose to use the percentage completed of the project approximately 1%. We chose percent completed because it would allow us to normalize all projects in the dataset, no matter the life span of the project or how many sprints/releases each project had.

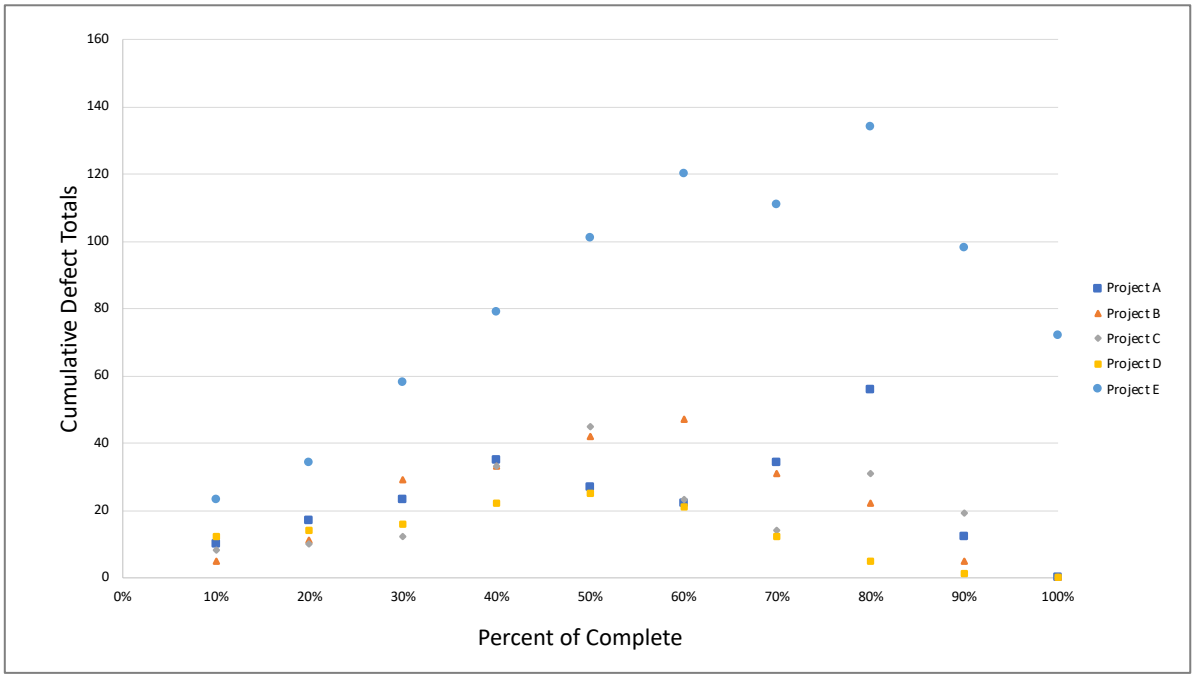


Fig.3 - Example of cumulative defects over normalized project lifespan by project

Next, we performed the same analysis on all projects in DevComp's dataset. Lastly, we visualized the outputs of each project using the scatterplot of cumulative defect costs for all projects, as illustrated in Figure 3. The plot can be used by the stakeholders of the projects to visualize the state of individual projects in the organization.

3.4. Prediction Technique & Model

As described in earlier sections we needed a technique to predict the outcome of the project early. This type of problem lends itself to using a machine learning (ML) algorithm. In order to determine whether a project is a success or failure based on the TD or our case defects, we had to use a classifier. We ended up choosing SVM algorithm for this research because of SVM's ability to stay relatively accurate with a smaller dataset (Basavanhally et al., 2015) Python was used for the transforming and data input, as well as the SVM algorithm. We used a collection of python libraries for this custom Extract, Transform, and Loading (ETL) process. NumPy (NumPy.org) was also used for parts of the transformation algorithm. Scikit-learn (Boisberranger, 2007) was used for the implementation of the SVM algorithm. For each project, the data created in section 3.3 was put into a dynamic list object.

Each list item contains relative completion time of the project (e.g. 0.1 or 10% of the expected project duration) and the cumulative defect count at that point in time, for each project in the

training dataset with 1% increment in the project's completion. So, each project in our dataset should have 100 data points in the list. We took an approach like that of the k-fold cross validation technique. We trained the model on all completed projects other than the one performing the prediction on. The dynamic list was then used as the training data to train the SVM model. We used a binomial classification technique for each data point in the list. It would correspond to the success or failure of the project (SUCCESS = 1, FAILURE = 0). If the project succeeded (defined by DevComp process), all data points or support vectors in the list were assigned a 1 (SUCCESS) for the label. Otherwise, all support vectors in the list were assigned a 0 (FAILURE) for the label. The labels were placed in a separate list and used later. In order to train a model for the SVM algorithm, we used both the training data defined in this section and the labeled data as inputs in the train function of the SVM model function in Scikit-learn. For our SVM model, we used the following parameters from Table 1.

Parameter Name	Parameter Value
Kernel Type	Dot
Kernel Cache	200
C	1.0
Convergence Epsilon	0.001
Max Iterations	100000
L Pos	1
L Neg	1
Epsilon	0.0
Epsilon Plus	0.0
Epsilon Minus	0.0

Table 1- SVM Model Properties

In order to perform the prediction analysis, we had to identify in our project list which projects were successes and which projects were failures. DevComp had a clear definition for success in their process so we used it for our labeling. We used their project outcome definition for labels for each project as defined by their process. We then defined which projects needed to be used for training data and which project we were going to do the predictions on. We used a project configuration file for this project. We defined this configuration in order to quickly change what projects we were using for training data and which projects we were performing prediction analysis on. The success property was used to define the color of the support vectors (e.g. true = green, false = red, maybe = yellow). The prediction type property would be used to direct the algorithm on which datasets to use for training data and which ones to use for prediction analysis. In a real-world scenario, you would set success = maybe and prediction type = predict on any real project that is ongoing in order to perform predictive analysis on. Each one of the labels is a guess by the prediction algorithm on what the outcome of the project will be based on the training data, for each data point in the project. The labels given by the prediction algorithm is as follows: true = success for the project and false = failure for the project. These data points can be aligned with the other support vectors in order to give a depiction of how the algorithm was predicting based on the debt at that point in the project. After the model was trained, we would use the trained model and a set of support vectors in an ongoing project to determine its binomial classification (SUCCESS = 1, FAILURE = 0) and a likelihood of the project succeeding or failing from the output of the SVM model.

4. Results and Discussion

In this section, we will discuss the results of the sections and research described earlier. We applied these algorithms and ideas to the DevComp JIRA issue database for about 6 months. The results are described below. Please note that all source code can be viewed at *https://github.com/tshrove/technicaldebtprediction*

4.1. Identification & Measurement Results

After DevComp allowed us access to their JIRA API, we were able to apply the algorithms discussed in this paper to their data. The first task we performed was to choose an existing project. We decided on a project that was currently in progress that we refer to as Proj M, to see what the TD would look like for an on-going project. We decided to only look at defects for this project and develop a graph like Figure 3 but using 100 support vectors per project as described in Section 3.3 instead of 10 as depicted in Figure 3.

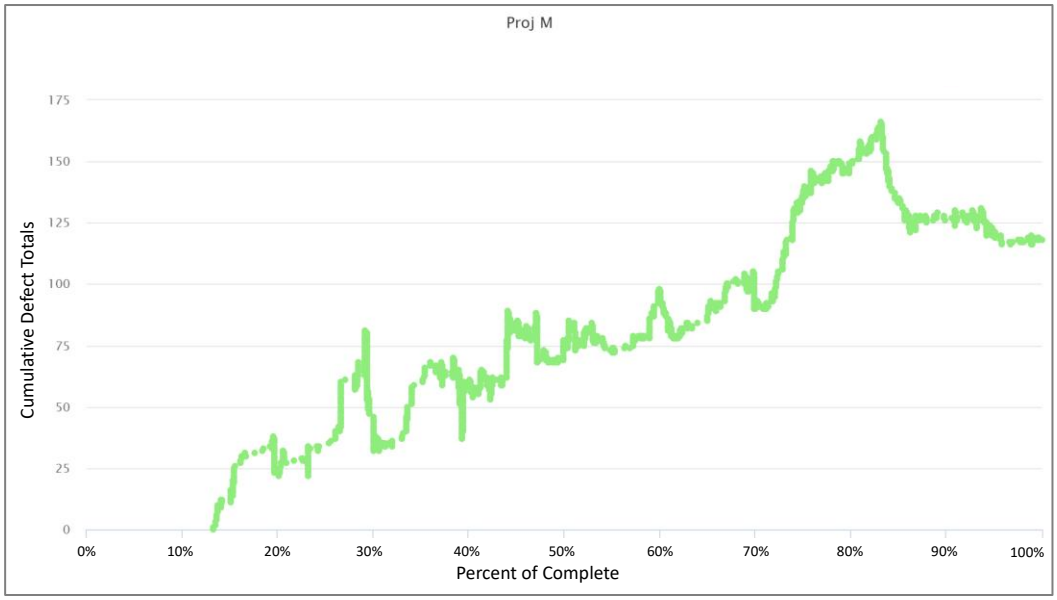


Fig.4 - ProjM cumulative defect totals over time

Figure 4 shows evolution of the technical debt for Proj M during project's life cycle. This project's defect costs started to increase around 15% of the project's expected life cycle. A good hypothesis would be that the project's testing of the first minor release would start around that time.

To compare all the projects together with respect to the outcome (success or failure), we plotted defects of all projects on one graph, as shown in Figure 5. We color coded the successful projects green, failure projects red, and on-going projects yellow in order to identify them easily on the scatterplot as seen in Figure 5. Ongoing projects are always represented as 100% complete at the

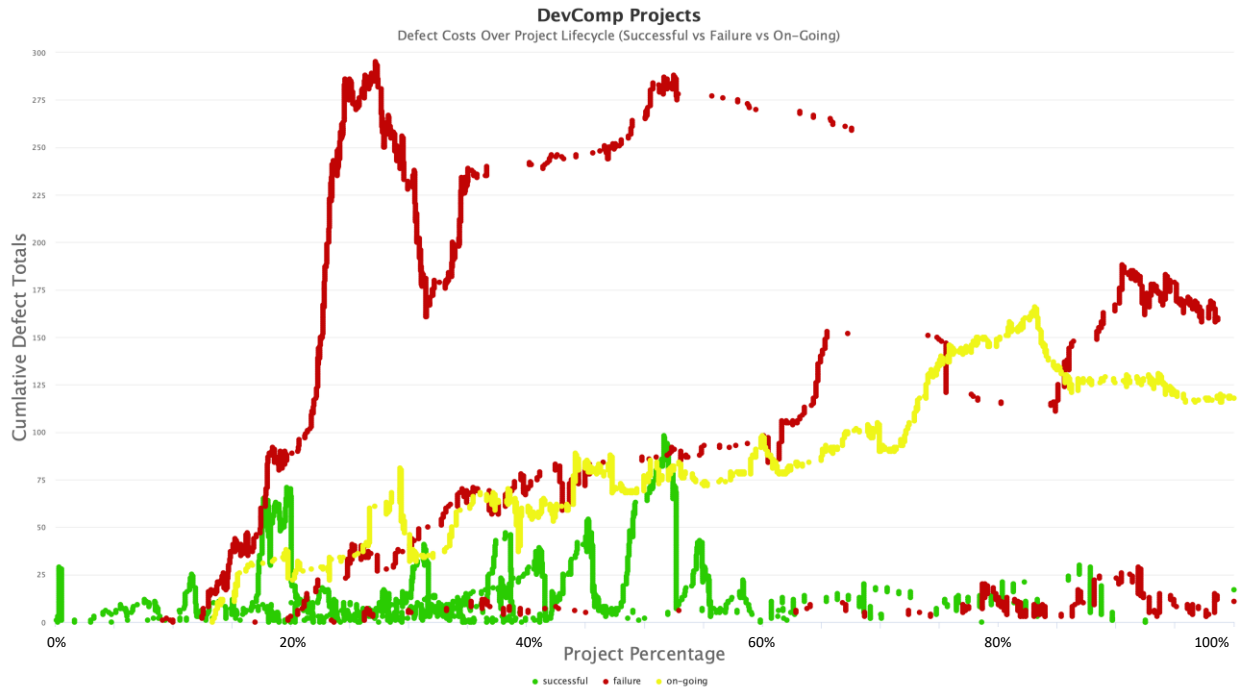


Fig.5 - Cumulative defect count of successful, failed, and on-going projects of DevComp

time of analysis. In presenting Figure 5, we are showing that successful (green) projects are tend to cluster together under the 100 mark of defect costs while unsuccessful project tend to trend upward and stay above 100. Please note that the one on-going project (yellow) had a trend toward the unsuccessful (red) project in the plot.

4.2. Results Validation

In order to analyze the performance or the prediction error of the model developed, we used a k-fold cross-validation method. We used the following steps to perform the performance tests (Fushiki, 2011):

1) We consolidated all data points into one set.
2) Split the dataset into k groups (in this case by project the data point was associated with)
3) For each dataset group
   a) Took the groups data as test data of the performance metric
   b) Took the remaining data as the training date for the model
   c) Evaluated the model against the test group data.
   d) Retained the evaluated performance data and discarded the model

Dataset of each group was used for testing once and for the model training k-1 times during the duration of the performance test. The variable k in our example is the number of completed projects in DevComp's portfolio. The results of the testing indicate error rate of 0.113 or 11.3%. We also

constructed a Receiver Operating Characteristic (ROC) curve to show and diagnose how well our binary classification methods actual performed. Figure 6 shows our ROC curve for the proposed classifier.

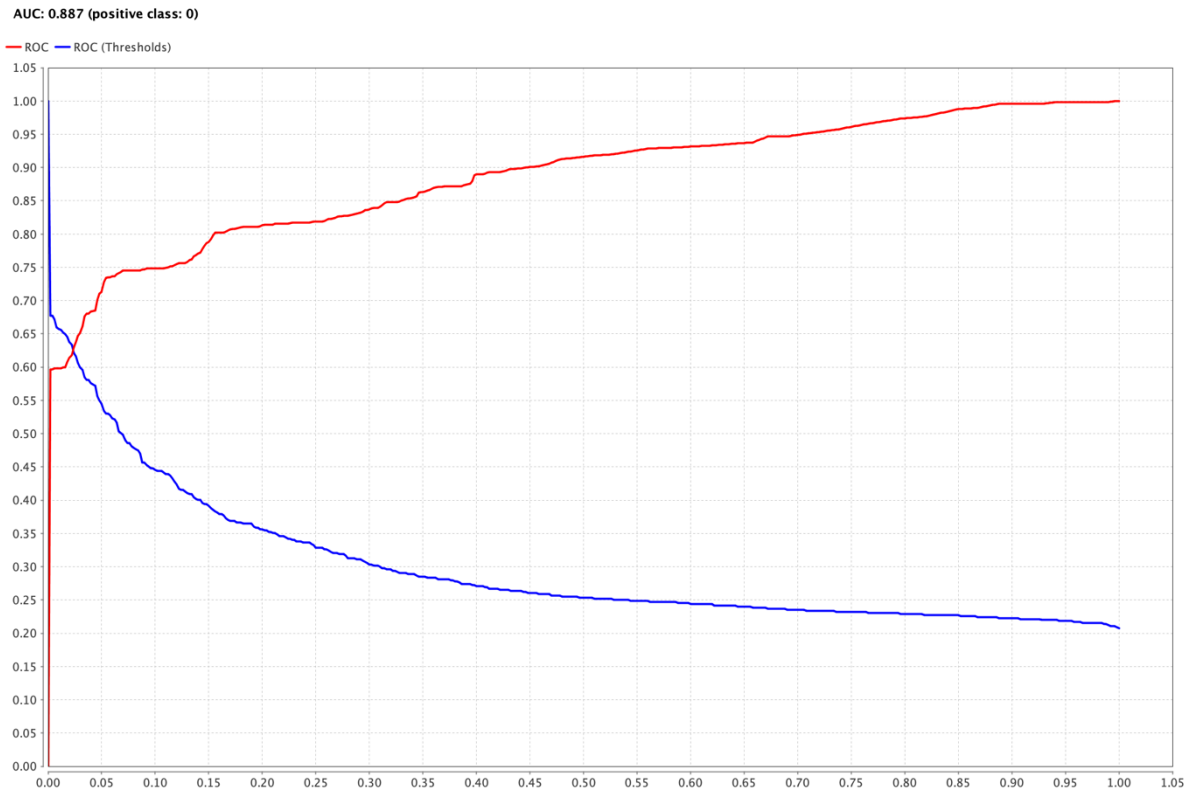


Fig.6 - ROC curve of prediction algorithm

5. Threats to Validity

In this section we discuss the limitations of our research and propose mitigation strategies for each potential threat.

5.1. Threat #1: Failure assessment

During our analysis we had to rely on DevComp's process and their assessment how successful the project s was. We labeled the project according to their assessment. In order to ensure that each project is getting a consistent label each time, we would need a formal definition of what failure is related to their process and the projects. That was not available to us during our research and dealings with DevComp. They considered that sensitive information and would not allow us access to that information.

However, the reader should note that the clear definition of success or failure that was not provided to us does not necessarily prevent our analysis. Since the assessment depends on DevComp's criteria, the assessment will be relevant to DevComp needs, with potentially limited usability for other developers. However, if each business or entity were to define their success criteria, they themselves could develop their own unique model using the same methods in previous sections but with new labels on each project. The model developed during this research should not be considered representative model of all soft-ware projects.

5.2. Threat #2: Data Validity

The defects collected in our data are defects reported by people in DevComp. These people may be amateurs to the product, or they could be new personnel. We don't really have any knowing, therefore, the reported defects may not be valid or there may be duplicates of the same defect.

5.3. Threat #3: Reporting Mechanism

The data collected for this research was gathered on DevComp's JIRA repository, however, DevComp could have another internal reporting instance of JIRA shown to our team. This would not reflect in our data or model development. In future research, we could reach verify this with data sources to confirm one instance of JIRA for defect reporting.

6. Conclusion

Software organizations need to continuously monitor software projects and identify as early as possible in their lifecycle if they would eventually fail. In this research, we used defects and a novel predictive model to monitor projects during their lifecycle. We introduced a novel method to calculate defect costs for features into an SVM model that can be used to predict likelihood of success of agile software projects.

Analysis of the set of real software projects indicate that the proposed method can effectively predict the outcome of the software project with accuracy of 88.7%. We believe that the model can be further improved using more data and additional features into the model (i.e. TD). We also plan to use additional dimensions in the SVM, such as time of the event, to make the prediction model more accurate.

7. References

Vashisht, V., Lal, M., & Sureshchandar, G. S. (2015). A Framework for Software Defect Prediction Using Neural Networks. *Journal of Software Engineering and Applications*, *08*(08), 384–394. https://doi.org/10.4236/jsea.2015.88038

Qinbao Song, Zihan Jia, Shepperd, M., Shi Ying, & Jin Liu. (2011). A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering, 37(3), 356–370. https://doi.org/10.1109/TSE.2010.90

Fernandez-Sanchez, C., Garbajosa, J., Vidal, C., & Yague, A. (2015). An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture Perspective. 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, 22–28. https://doi.org/10.1109/SAM.2015.11

Ramaswamy, V., Suma, V., & Pushphavathi, T. (2012). An approach to predict software project success by cascading clustering and classification. IET Seminar Digest, 2012(4). https://doi.org/10.1049/ic.2012.0137

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 34(4), 485–496. https://doi.org/10.1109/TSE.2008.35

Bou-Hamad, I., & Jamali, I. (2020). Forecasting financial time-series using data mining models: A simulation study. Research in International Business and Finance, 51. https://doi.org/10.1016/j.ribaf.2019.101072

Nam, J., Fu, W., Kim, S., Menzies, T., & Tan, L. (2018). Heterogeneous Defect Prediction. IEEE Transactions on Software Engineering, 44(9), 874–896. https://doi.org/10.1109/TSE.2017.2720603

PyContribs. (2017). JIRA Python Library. Retrieved October 2, 2018, from https://github.com/pycontribs/jira

Akbarinasaji, S., Bener, A., & Erdem, A. (2016). Measuring the principal of defect debt. Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, 1–7. https://doi.org/10.1145/2896995.2896999

NumPy.org. (n.d.). NumPy — NumPy. Retrieved October 2, 2018, from http://www.numpy.org/

Lehtinen, T., Mäntylä, M., Vanhanen, J., Itkonen, J., & Lassenius, C. (2014). Perceived causes of software project failures – An analysis of their relationships. Information and Software Technology, 56(6), 623–643. https://doi.org/10.1016/j.infsof.2014.01.015

Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., & Mishra, R. (2007). Predicting software defects in varying development lifecycles using Bayesian nets. Information and Software Technology, 49(1), 32–43. https://doi.org/10.1016/j.infsof.2006.09.001

Weber, R., Waller, M., Verner, J., & Evanco, W. (2003). Predicting software development project outcomes. Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2689, 595–609. https://doi.org/10.1007/3-540-45006-8_45

Boisberranger, J. Du, Bossche, J. Van den, & et. al. (2007). scikit-learn: machine learning in Python — scikit-learn 0.20.0 documentation. Retrieved October 2, 2018, from http://scikit-learn.org/stable/

Okutan, A., & Yıldız, O. (2014). Software defect prediction using Bayesian networks. Empirical Software Engineering, 19(1), 154–181. https://doi.org/10.1007/s10664-012-9218-8

Wojewoda, S., & Hastie, S. (2015). Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. Retrieved August 25, 2019, from https://www.infoq.com/articles/standish-chaos-2015/

Shuo Wang, & Xin Yao. (2013). Using Class Imbalance Learning for Software Defect Prediction. IEEE Transactions on Reliability, 62(2), 434–443. https://doi.org/10.1109/TR.2013.2259203

Fushiki, T. (2011). Estimation of prediction error by using K -fold cross-validation. Statistics and Computing, 21(2), 137–146. https://doi.org/10.1007/s11222-009-9153-8

Basavanhally, A., Viswanath, S., Madabhushi, A., & Basavanhally, A. (2015). Predicting classifier performance with limited training data: applications to computer-aided diagnosis in breast and prostate cancer. PloS One, 10(5), e0117900–e0117900. https://doi.org/10.1371/journal.pone.0117900

Guillaume-Joseph, G., Wasek, J., Blessner, P., Mazzuchi, T., Murphree, E., Sarkani, S., & Wasek, J. (2016). Improving Software Project Outcomes through Predictive Analytics (ProQuest Dissertations Publishing). Retrieved from http://search.proquest.com/docview/1752636196/