P1 - Project Statement
CSCI 6509 - Natural Language Processing

March 12, 2019

Jadeja Nirav - B00789139
Patel Supriya - B00791627
Singh Abhishek - B00782673

Contents

1 Project Title

Self-Admitted Technical Debt Detection Using NLP.

2 Problem Statement

The term Technical Debt (TD) refers to a number of shortcuts taken by developers to complete the assigned work in time or to be the solitary one in the market [1]. From the literature, it was perceived that approximately 75% of developers were not even familiar with the term though they admitted to performing TD on a regular basis when explained a generic definition. The concern with this is there are so many forms of TD that even developers may not be aware of and this only leads to accumulation of re-factoring work in the future. Among the several types of TD such as Design, Architectural, Bit Rot, Accidental, Requirement and so on. We aim to focus on design and requirement debt. TD due to requirement debt typically occurs when developers do not understand the requirements accurately and instead of clarifying them, they make assumptions which in turn lead to additional work in future. Mistakes like these can be overcome easily if detected on time. Moreover, from time to time the structure of the code is defined as unstable or fragile which happens when the entire code is written by several developers and then stitched together haphazardly instead of a properly designed code [2]. Hence, the problem we aim to tackle in this project is the detection of requirement and design TD that is Self-Admitted by developers themselves (i.e. not accidental) which would, in turn, prevent future costs and time, not to mention increased efforts as well.

3 Possible Approaches

Few approaches have surfaced regarding the detection of Self-Admitted Technical Debt (SATD). It has a rather moderate amount of literature review to study distinct techniques for the detection of SATD. Previously, automated tools have been used to detect TD, but they are able to perceive eminently defect debt [3]. Using NLP allows us to focus on keywords in source code comments and assign them weights to classify them in categories. Apart from using source code comments, static source code analysis, which is also known as code smells can be opted for though using source code comments approach is more lightweight and inexpensive [4] [5]. One of the possible approaches to NLP classification is manual classification. Detecting different types of debts may prove difficult because we

cannot assign weights and train the model to categorize in manual classification, but we also have an advantage of not necessitating plentiful data. For NLP classifier to provide high accuracy we require sufficient data for training. We can use Deep Neural Network (DNN) for even higher accuracy but some of the questions which arise are, Will DNN have a significant effect on accuracy? Do we require that accuracy for effective classification? Another aspect taking part in the selection of a classifier is whether we are aiming for recall or precision. Naive Bayes classifiers are comparatively simple to use and by opting to observe false positives we can study why comments are classified falsely but if we aim for accuracy then precision matters, for which we use regression classifiers [1].

4 Project Plan

At present, we are only targeting Java based open-source projects. For the entire project, we would utilize Python and related libraries (i.e TensorFlow, Keras). Further, the project plan for the rest of the term (4 weeks) is divided into the following steps:

- Week 1: We start by selecting 10 open source projects with varying amount of source code comments which is necessary for standard evaluations. Once we finalize our projects, we extract source code comments using JDeodrant which is an open source plugin. This is quite a useful plugin which also allows us to detect bad smells. Next using filtering heuristics, we remove irrelevant comments and only keep the once which hint towards SATD and create a dataset.

- Week 2: Once the dataset is ready, we will start understanding the words which represent specific comments such as design debt and requirement debt have separate comment keywords through which we can classify them accurately. Previous approaches have carried out an exercise to classify these comments manually to compare later on [6][7]. We will be using this manually classified data for our comparison purposes as well. Pre-processing will be done on this dataset to remove repeated words or comments with differing letter cases as well as character cases, punctuation, etc. which may classify as redundant data.

- Week 3: Using this dataset, we will prepare a model with maximum entropy classifier. Considerations such as overfitting will be taken into perspective and techniques like K-fold cross-validation, leave-one-out cross-validation will be performed according to the size of the data. Similar models would be studied to find out the difference

between the results to see if we can find an optimal model. The effectiveness of this NLP model would be determined by comparing the F1 measure with the approaches in the literature review [8].

- Week 4: Determining through analyzing and implementation how strict we want our parameters for Source Line of Codes (SLOC), filtering heuristics, classifiers and training data size.

References

[1] E. S. Maldonado, E. Shihab, and N. Tsantalis, “Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt“, vol. XX, no. X, pp. 1-20, 2017.

[2] E. Lim and A. Informatics, “A Balancing Act : What Software Practitioners Have to Say about Technical Debt“, pp. 22-28.

[3] N. Zazworka, R. O. Spinola, A. Vetro, F. Shull, and C. Seaman, “A case study on effectively identifying technical debt”, Int. Conf. Eval. Assess. Softw. Eng., pp. 42-47, 2013.

[4] N. Tsantalis and A. Chatzigeogiou, “Identification of extract method refactoring opportunities for the decomposition of methods“, J. Syst. Softw., vol. 84, no. 10, pp. 1757-1782.

[5] N. Tsantalis, D. Mazinanian, and G. P. Krishnan, “Assessing the refactorability of software clones“, IEEE Trans. Softw. Eng., vol. 41, no. 11, pp. 1055-1090, 2015.

[6] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. SpÃŋnola, “Towards an Ontology of Terms on Technical Debt“, pp. 1-7, 2014.

[7] E. Maldonado, E. Shihab, and N. Tsantalis, “Replication Package for Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt“, 2016.

[8] A. Potdar, “An Exploratory Study on Self-Admitted Technical Debt“, 2014.