UNIVERSITÉ DE MONTRÉAL

RECOMMENDING WHEN DESIGN TECHNICAL DEBT SHOULD BE SELF-ADMITTED

CÉDRIC NOISEUX
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AÔUT 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

RECOMMENDING WHEN DESIGN TECHNICAL DEBT SHOULD BE SELF-ADMITTED

présenté par: NOISEUX Cédric
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. NOM Prénom, Doct., président
Mme NOM Prénom, Ph. D., membre et directrice de recherche
M. NOM Prénom, Ph. D., membre

DEDICATION

À tous mes amis du labos,
vous me manquerez. . . FACULTATIF

ACKNOWLEDGEMENTS

Texte. FACULTATIF

RÉSUMÉ

TOTAL = 4 pages

Technical debts (TD) are temporary solutions, or workarounds, introduced in portions of software systems in order to fix a problem rapidly at the expense of quality. Such practices are widespread for various reasons: rapidity of implementation, initial conception of components, lack of system's knowledge, developer inexperience or deadline pressure. Even though technical debts can be useful on a short term basis, they can be excessively damaging and time consuming in the long run. Indeed, the time required to fix problems and design code is frequently not compatible with the development life cycle of a project. This is why the issue has been tackled in various studies, specifically in the aim of detecting these harmful debts.

One recent and popular approach is to identify technical debts which are self-admitted. The particularity of these debts, in comparison to TD, is that they are explicitly documented with comments and intentionally introduced in the source code. SATD are not uncommon in software projects and have already been extensively studied concerning their diffusion, impact on software quality, criticality, evolution and actors. Various detection methods are currently used to identify SATD but are still subject to improvement. For example, using keywords (*e.g.: hack, fixme, todo, ugly, etc.*) in comments linking to a technical debt or using Natural Language Processing (NLP) in addition to machine learners. Therefore, this study investigates to what extent previously self-admitted technical debts can be used to provide recommendations to developers writing new source code. The goal is to be able to suggest when to "self-admit" technical debts or when to improve new code being written.

To achieve this goal, a machine learning approach was conceived, named TEDIOUS (TEchnical Debt IdentificatiOn System), using various types of method-level input features as independent variables to classify design technical debts using self-admitted technical debts as oracle. The model was trained and assessed on nine open source java projects which contained previously tagged SATD. In other words, our proposed machine learning approach aims to accurately predict technical debts in software projects.

TEDIOUS works at method-level granularity, in other words, it can detect whether a method contains a design debt or not. It was designed this way because developers are more likely to self-admit technical debt for methods or block. TD can be classified in different types: design, requirement, test, code or documentation. Only design debts were considered because they represent the largest fraction and each type would require its own analysis. TEDIOUS is trained with *labeled data*, projects with labeled SATD, and tested with *unlabeled data*.

The labeled data contain methods tagged as SATD which were obtained from nine projects manually analyzed by another research group. From the labeled data are extracted various kinds of metrics: source code metrics, readbility metrics and warnings raised by static analysis tools. Nine source code metrics were retained to capture the size, coupling, complexity and number of components in methods. The readability metric takes in consideration indentation, line length and identifier lengths just to name a few features. Two static analysis tools are used to check for poor coding practices.

Feature preprocessing is applied to remove unnecessary features and only keep h

ABSTRACT

TOTAL = 4 pages

VOIR PROPOSITION DE RECHERCHER

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATION

IETF	Internet Engineering Task Force
OSI	Open Systems Interconnection

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

TOTAL = 8 pages

0.5 page

Base sur abstract de l'article VOIR PROPOSITION DE RECHERCHE

1.1 Basic Concepts and Defintions

environ 3 pages

Technical debt Defintion TD (Cunningham 15) Definition classes TD (3 et 22) Nature intentionnelle des TD (24) Awareness des TD est un probleme (17)

Self admitted technical debt Definition SATD (Potdar 16 et 35) Presence des SATD (Potdar 16 et 35) Acteurs des SATD (Potdar 16 et 35) Correlation qualite/SATD (Bavota et Russo 8, 10) Taxonomie des SATD (Bavota et Russo 8)

Features Structural metrics of methods Method readability Warnings of static analysis tools (Checkstyle 1, PMD 2)

Dataset 9 java open source projects from (27), only design debts

Machine Learners 5 machine learners, with fold validation and cross project

Results 50% precision and 52% recall within project with RandomForest 67% precision, 55% recall, 92% accuracy cross project Can be applied on new projects and be good

1.2 Éléments de la problématique

environ 3 pages

1.3 Objectifs de recherche

0.5 page

TEDIOUS (TEchnical Debt IdentificatiOn System) Supervised Machine Learning approach Method-level Using various features of code source (independant) Knowledge of previous SATD (dependant) To recommend developpers with TD to be admitted

Purpose 1) Encourage self admitting TD (mainly done by experienced, want new to do

too 35) 2) Alternative to smell detectors to give opportunities to improve source code

Difference 1) Method-level instead of class-level metrics (8) because SATD comments at method or block-level mainly 2) Only consider certain types of TD (design debt since largest 27) Other types for future since different analysis

1.4 Plan du mémoire

0.5 page

CHAPTER 2 LITTERATURE REVIEW

TOTAL = 4 pages

2.1 Relationship Between Technical Debt and Source Code Metrics

2.2 Self-Admitted Technical Debt

2.3 Code Smell Detection and Automated Static Analysis Tools

CHAPTER 3 THE APPROACH AND STUDY DEFINITION

TOTAL = 15 pages

3.1 The Approach

2 pages

3.1.1 Features

3 pages

3.1.2 Identification of Self-Admitted Technical Debt

0.5 page

3.1.3 Feature Preprocessing

3 pages

3.1.4 Building and Applying Machine Learning Models

0.5 page

3.2 Study Definition

0.5 page

3.2.1 Dataset

2 pages

3.2.2 Analysis Method

3.5 pages

CHAPTER 4 ANALYSIS OF STUDY RESULTS AND THREATS TO VALIDITY

TOTAL = 18 pages

4.1 Study Results

4.1.1 How does TEDIOUS work for recommending SATD within-project?

4.5 pages

4.1.2 How does TEDIOUS work for recommending SATD across-project?

4 pages

4.1.3 How would a method-level smell detector compare with TEDIOUS?

1.5 pages

4.1.4 Qualitative discussion of false positive and false negatives

4 pages

4.2 Threats to Validity

4.2.1 Construct validity

1 page

4.2.2 Internal validity

1 page

4.2.3 Conclusion validity

1 page

4.2.4 External validity

1 page

CHAPTER 5 CONVOLUTIONAL NEURAL NETWORK WITH COMMENTS AND SOURCE CODE

TOTAL = 15 pages

5.1 Convolutional Neural Network

1 page

5.2 The Approach

1 page

5.2.1 Features

1 page

5.2.2 Identification of Self-Admitted Technical Debt

0.5 page

5.2.3 Word Embeddings

1 page

5.2.4 Building and Applying CNN

2 page

5.3 Study Definition

0.5 page

5.3.1 Dataset

1.5 page

5.3.2 Analysis Method

2 pages

5.4 Study Results

5.4.1 Source Code With Comments

1.5 pages

5.4.2 Source Code Without Comments

1.5 pages

5.4.3 Source Code Partially With Comments

1.5 pages

CHAPTER 6 CONCLUSION

TOTAL = 3 pages

6.1 Summary of Work

1 page

6.2 Limitations of the Proposed Solution

1 page

6.3 Future Work

1 page

BIBLIOGRAPHY

APPENDIX A DÉMO

Texte de l'annexe A. Remarquez que la phrase précédente se termine par une lettre majuscule suivie d'un point. On indique explicitement cette situation à LaTeX afin que ce dernier ajuste correctement l'espacement entre le point final de la phrase et le début de la phrase suivante.

APPENDIX B ENCORE UNE ANNEXE

Texte de l'annexe B en mode «landscape».

APPENDIX C UNE DERNIÈRE ANNEXE

Texte de l'annexe C.