RECOMMENDING WHEN DESIGN TECHNICAL DEBT SHOULD BE SELF-ADMITTED

Computer Engineering

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Cedric Noiseux

December 5, 2017

Table of Contents I

Table of Contents II

Concepts and Definitions

Technical Debt (TD)

Definition

TDs are not quite right code which we postpone making it right [1].

They are temporary and suboptimal solutions or workarounds.

Reasons

Quickly fix an issue

Early conception stages

Lack of comprehension, skill or experience

Concepts and Definitions

Technical Debt (TD)

Types

Design	Object-Oriented design principles
Requirement	Functional and non functional requirement trade-offs
Code	Quality of source code
Test	Quality of testing activities
Documentation	Misleading Information
Defect	Known and possible defects

Impacts

Slower conception and execution

Lower maintainability and quality

Higher production cost

Concepts and Definitions

Self-Admitted Technical Debt (SATD)

Definition

SATDs are technical debts which are consciously introduced and explicitly documented by developers.

31% of files contain SATD

Persistent through time

Experienced developers introduce them

Elements of the Problematic

Self-Admitted Technical Debt (SATD)

Issues

Weak Performance Results

Ineffective Strategies

- Code Refactoring
- Continuous Tracking
- Proactiveness

Flawed or Incomplete Approaches

- Pattern Matching
- Machine Learning with Natural Language Processing

Objectives

Main Research Question

How can we identify and detect technical debts in a software project using source code or features and known self-admitted technical debts with a machine learning approach?

TEDIOUS	CNN
Metrics and Warnings	Source Code
Machine Learners	Neural Network

Objectives

Application Objectives

Recommendation System

To encourage self-admitting technical debts

To ease tracking

To fix issues faster

Complement to existing smell detectors

Or use as an alternative

Tracking **Managing** **Improving**

Objectives

Design Objectives

Defining and Extracting Features

TEDIOUS : Metrics and Warnings

CNN : Source Code

Identifying SATDs

Preprocessing Features

TEDIOUS : Multi-Collinearity, Feature Selection, Normalization, Rebalancing

CNN : Tokenization, Word Embedding

Building and Applying Machine Learners

TEDIOUS : Machine Learning Models

CNN : Neural Network

Research questions (main and 3 sub-questions) 9 OOS + Java More method-related than class-related Unbalance

Method-level Image du process (separer en differents process et expliquer) - Training and test sets - SATD matching - Features - Preprocessing - Building and applying machine learners (training and testing)

Analysis Method Within project performance Cross project performance TEDIOUS vs method level Qualitative analysis

Inspirations (Kim, Dos Santos) Description

Method-level Image du process (separer en differents process et expliquer) - Training and test sets - SATD matching - Source code - Preprocessing/Word Embedings - Building and applying machine learners (training and testing)

Research questions (main and 3 sub-questions) 9 OOS + Java More method-related than class-related Unbalance

Source code comments only Source code with comments Source code without comments Source code partially with comments Construct Internal Conclusion Reliability External Number of metrics/warning Number of LOC Process inherent errors Default configurations (no real optimizations) Generalization of results Only in Java Performance evaluation can be better for CNN TEDIOUS approach CNN approach Dataset TEDIOUS results CNN results Applicability scenarios - Recommendation system - Complement to smell detectors Improvements - Optimization - Pattern matching process - More examples to train - More positive examples - More metrics/warnings and source code - Other TD types - Extend to other languages and domains

Bibliography

TEX, LATEX, and Beamer

[1] W Cunningham. *The wycash portfolio management system*. dans Addendum to the Proceedings on Object-oriented Programming Systems, Languages, and Applications (Addendum), série OOPSLA ’92, New York, NY, USA: ACM, 1992, pp 29–30, DOI: 10.1145/157709.157715. En ligne: http://doi.acm.org/10.1145/157709.157715.