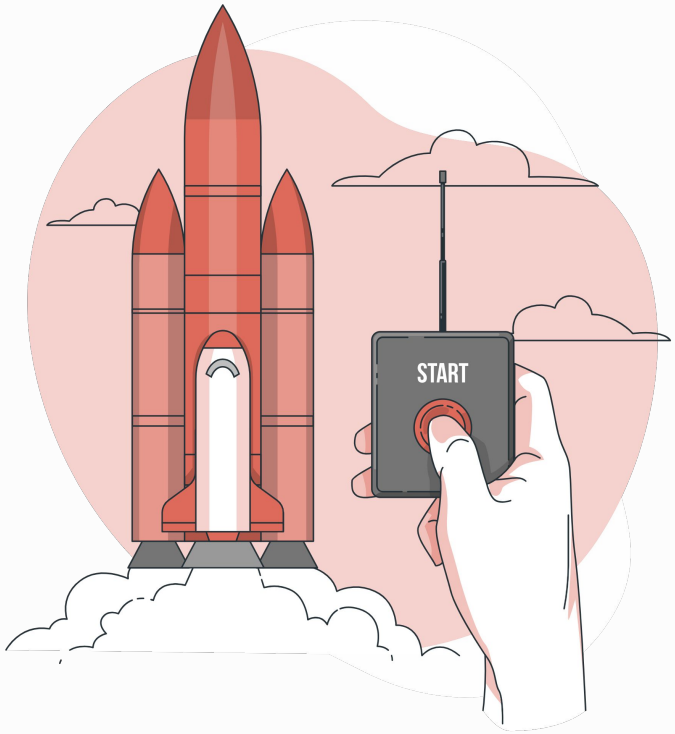
lab-soft
software engineering laboratory

Self-Admitted Technical Debt

Altino Alves Júnior

October 23rd, 2023



01

Introduction

What’s Technical Debt and SATD?

Technical Debt

Ward Cunningham (1992) first introduced the concept of considering the “not-quite-right code” as a form of debt. Technical debt is a metaphor introduced to describe the situation where long-term code quality is traded for short-term goals.

- However, **technical debt <u>is not always visible</u>**.

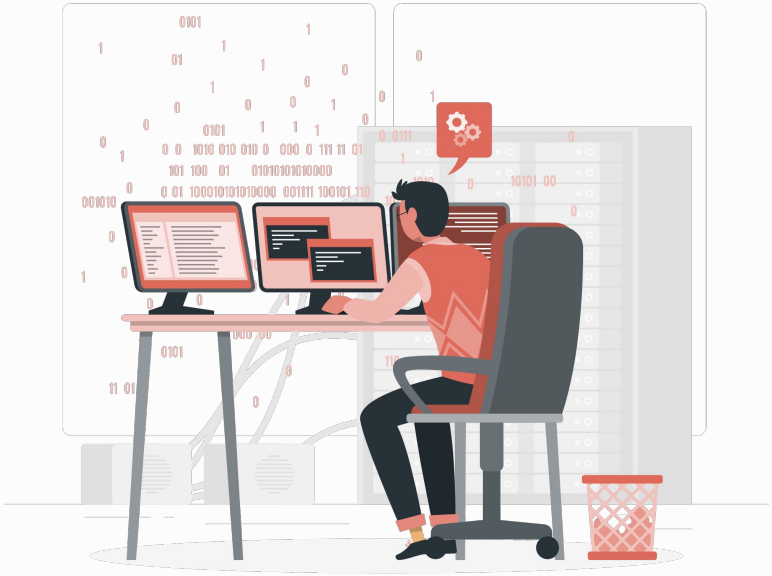
Cunningham, Ward. "The WyCash portfolio management system." ACM Sigplan Oops Messenger 4.2 (1992): 29-30.

Self-Admitted Technical Debt

Potdar and Shihab (2014) proposed the concept of self-admitted technical debt **(SATD)**, which considers debt that is intentionally introduced.

- i.e., Code that's either incomplete, defective, temporary or simply sub-optimal.

Developers document this using code comments or system messages.



A. Potdar and E. Shihab, "An Exploratory Study on Self-Admitted Technical Debt," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 2014, pp. 91-100, doi: 10.1109/ICSME.2014.31.

02

Detection

SATD detection methods and strategies

Detection

In the life cycle of **SATD**, debt instances are first introduced by developers into the source code. Thus naturally, the first step to study this phenomenon is to identify it.

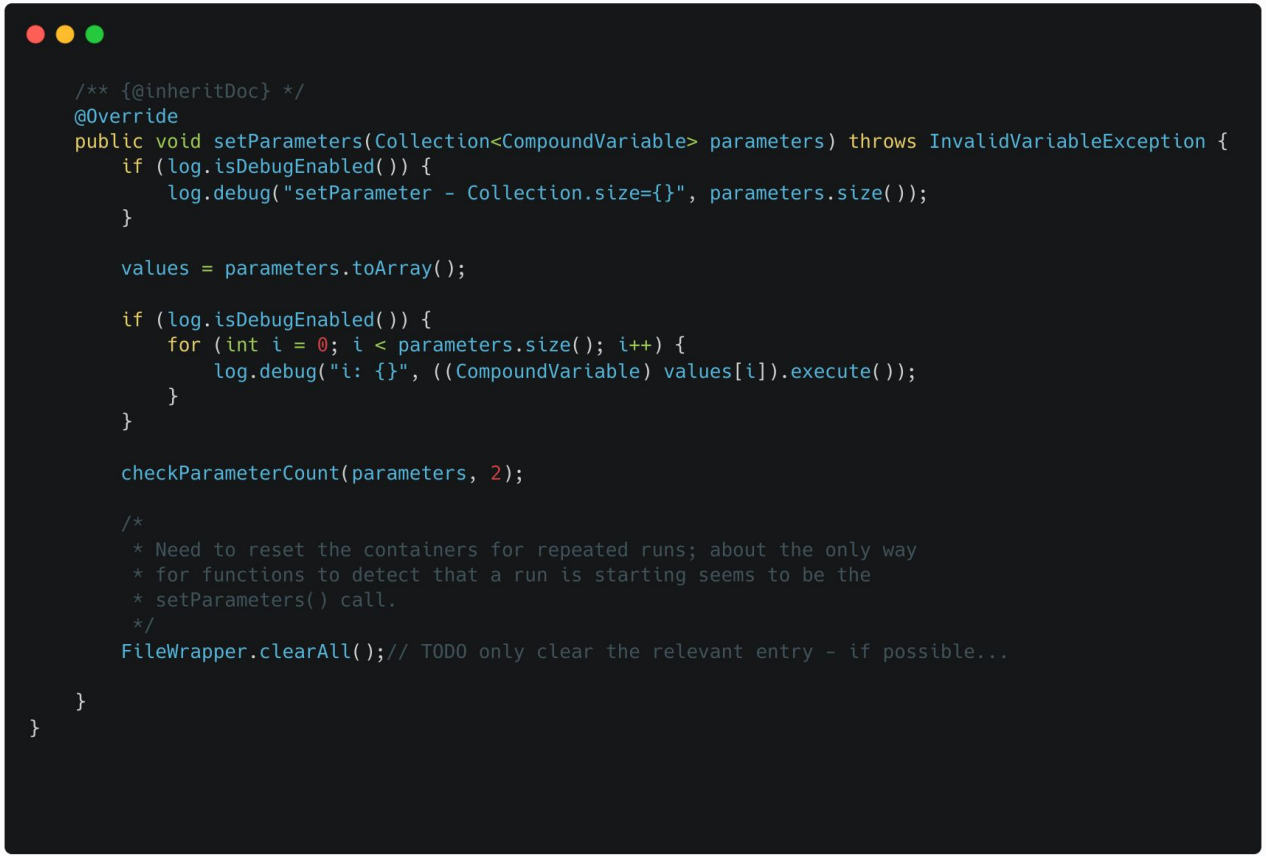
“TODO: - This method is too complex, lets break it up”

from ArgoUml

“Hack to allow entire URL to be provided in host field”

from JMeter

A. Potdar and E. Shihab, "An Exploratory Study on Self-Admitted Technical Debt," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 2014, pp. 91-100, doi: 10.1109/ICSME.2014.31.
E. d. S. Maldonado, E. Shihab and N. Tsantalis, "Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt," in IEEE Transactions on Software Engineering, vol. 43, no. 11, pp. 1044-1062, 1 Nov. 2017, doi: 10.1109/TSE.2017.2654244.



```
    /** {@inheritDoc} */
    @Override
    public void setParameters(Collection<CompoundVariable> parameters) throws InvalidVariableException {
        if (log.isDebugEnabled()) {
            log.debug("setParameter - Collection.size={}", parameters.size());
        }

        values = parameters.toArray();

        if (log.isDebugEnabled()) {
            for (int i = 0; i < parameters.size(); i++) {
                log.debug("i: {}", ((CompoundVariable) values[i]).execute());
            }
        }

        checkParameterCount(parameters, 2);

        /*
         * Need to reset the containers for repeated runs; about the only way
         * for functions to detect that a run is starting seems to be the
         * setParameters() call.
         */
        FileWrapper.clearAll();// TODO only clear the relevant entry - if possible...

    }
}
```

from JMeter | CSVRead.java File - Edited with Carbon

Detection Strategies

Pattern-based approaches

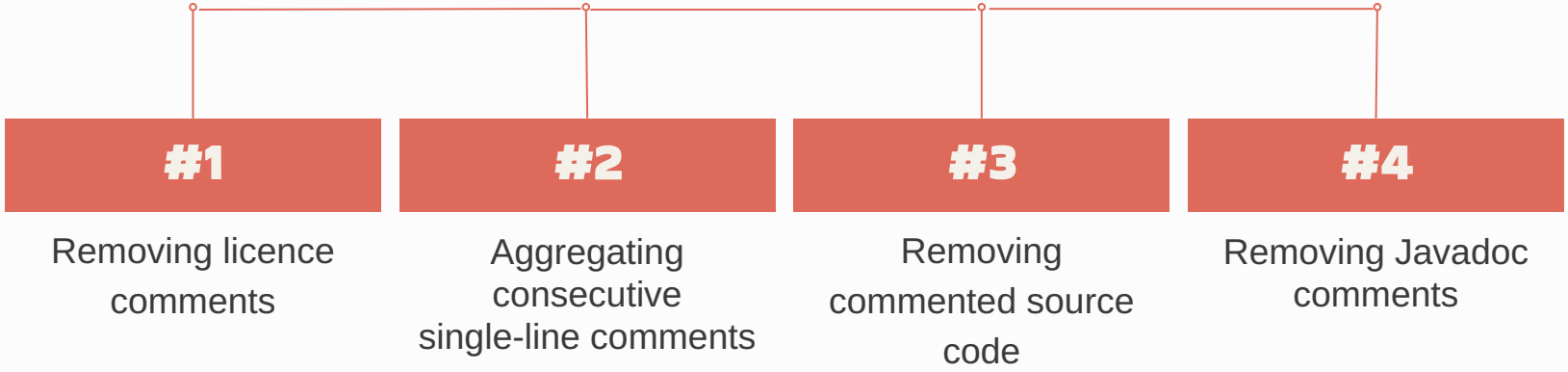
One of the main study about, Potdar and Shihab (2014) finded **62 patterns** and made them publicly available to enable further research.

- Some examples are: hack, fixme, is problematic, probably a bug, hope everything will work, etc.

A. Potdar and E. Shihab, "An Exploratory Study on Self-Admitted Technical Debt," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 2014, pp. 91-100, doi: 10.1109/ICSME.2014.31.

Detection Strategies

SATD filtering heuristics



A. Potdar and E. Shihab, "An Exploratory Study on Self-Admitted Technical Debt," 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 2014, pp. 91-100, doi: 10.1109/ICSME.2014.31.
E. d. S. Maldonado, E. Shihab and N. Tsantalis, "Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt," in IEEE Transactions on Software Engineering, vol. 43, no. 11, pp. 1044-1062, 1 Nov. 2017, doi: 10.1109/TSE.2017.2654244.

Detection Strategies

Pattern-based approaches

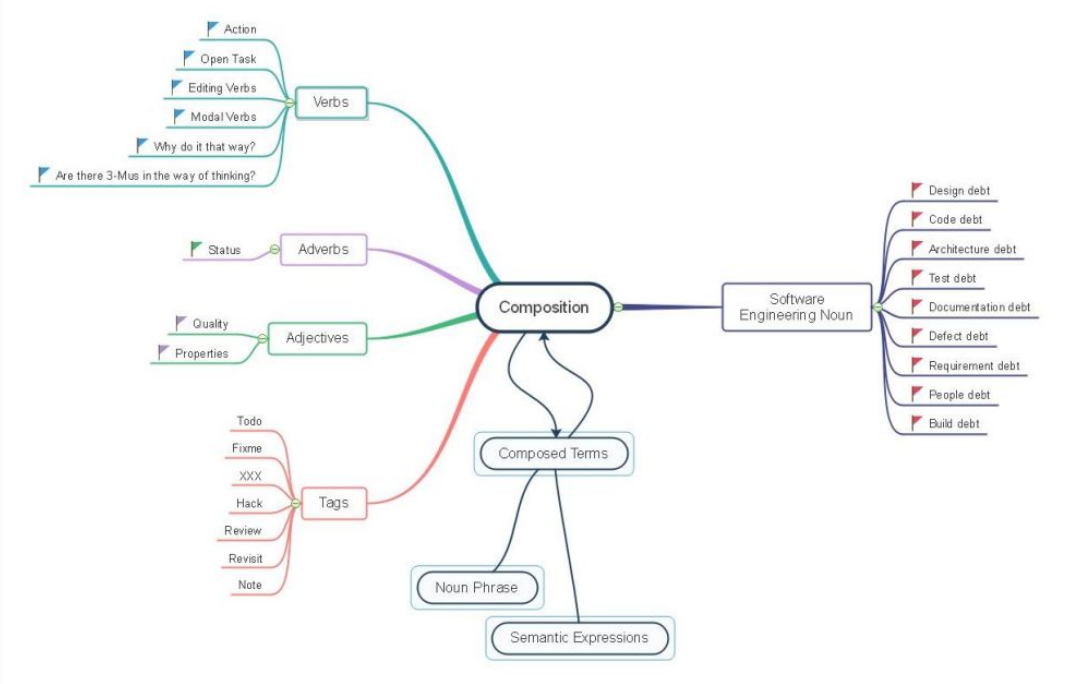
Contextualized Vocabulary Model

An alternative and extension to the pattern-based detection approach was later proposed by de Freitas Farias et al. (2015), who introduced CVM-TD for Identifying TD of different types in source code comments.

This model relies on identifying word classes, namely: **nouns**, **verbs**, **adverbs**, and **adjectives** that are related to Software Engineering terms and code tags.

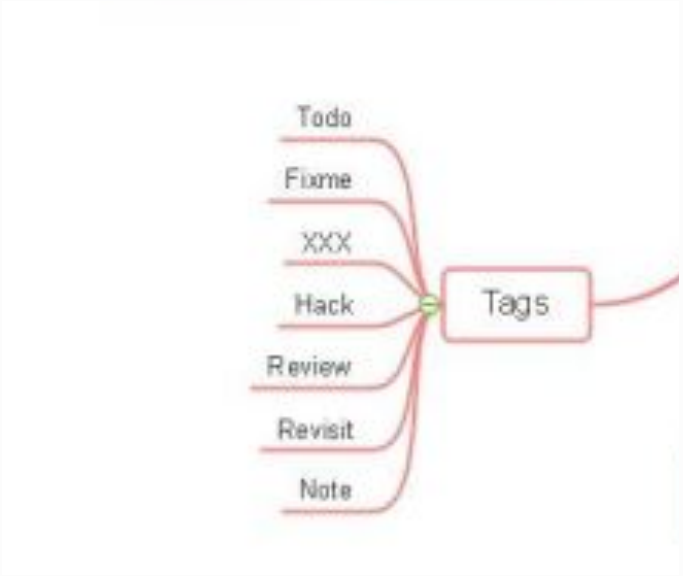
M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. d. Silva and R. O. Spínola, "A Contextualized Vocabulary Model for identifying technical debt on code comments," 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Bremen, Germany, 2015, pp. 25-32, doi: 10.1109/MTD.2015.7332621.

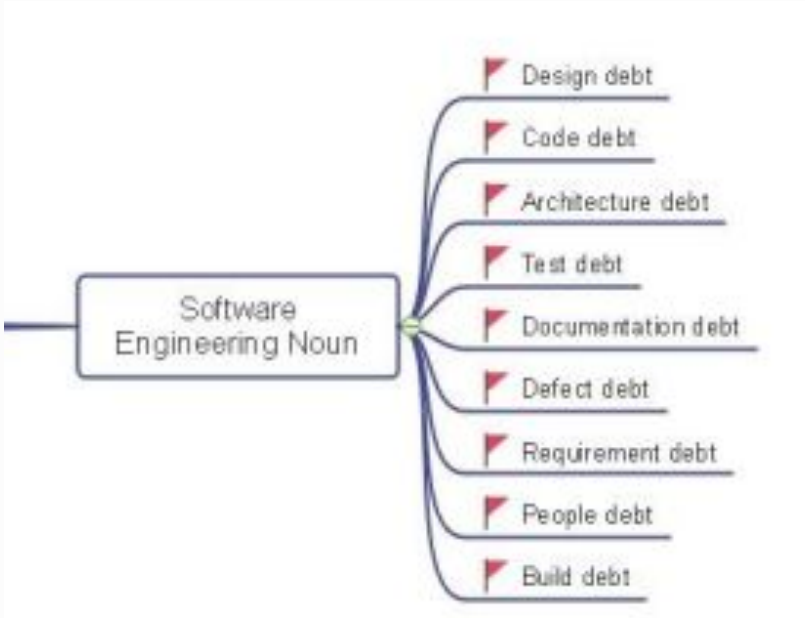
Contextualized Vocabulary Model



M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. d. Silva and R. O. Spínola, "A Contextualized Vocabulary Model for identifying technical debt on code comments," 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Bremen, Germany, 2015, pp. 25-32, doi: 10.1109/MTD.2015.7332621.

Contextualized Vocabulary Model





M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. d. Silva and R. O. Spínola, "A Contextualized Vocabulary Model for identifying technical debt on code comments," 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Bremen, Germany, 2015, pp. 25-32, doi: 10.1109/MTD.2015.7332621.

Text Mining

The process of exploring, analyzing and transforming large amounts of unstructured text data aided by software that can identify **concepts**, **patterns**, **topics**, **keywords** and other interesting **attributes** in the data.

Doing so typically involves the use of **natural language processing** (NLP) technology, which applies computational linguistics principles to parse and interpret data sets.

Detection Strategies

Machine learning approaches

Natural Processing Language (NLP)

Refers to the branch of computer science, artificial intelligence – concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics with **statistical**, **machine learning** and **deep learning models**.

Detection Strategies

Machine learning approaches

Huang et al. (2020) proposed an approach to **automatically** detect SATD using text mining and a composite classifier, named **Ensemble text mining** approach. It's root concept is to determine if a comment indicates SATD or not based on training comments from different software projects.

- This approach preprocesses comments by **tokenizing**, **removing stop-words** and **stemming their descriptions** to obtain textual features.

Zhongxin Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping L. SATD detector: a text-mining-based self-admitted technical debt detection tool. In Proceedings of International Conference on Software Engineering: Companion Proceeedings (ICSE). Association for Computing Machinery, New York, NY, USA, 9–12. https://doi.org/10.1145/3183440.3183478

03

Results

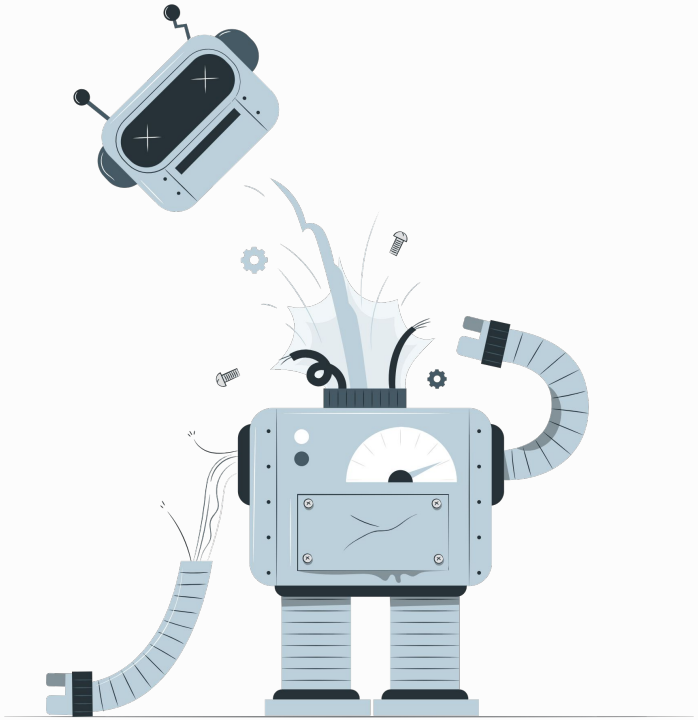
Comprehension, Impact and Future Work

Comprehension

Types of SATD

	Example	Project
Design Debt	“/*TODO: really should be a separate class */”	ArgoUml
Defect Debt	“Bug in the above method”	Apache JMeter
Requirement Debt	“//TODO no methods yet for getClassname”	Apache Ant
Documentation Debt	“**FIXME** This function needs documentation”	Columba
Test Debt	“//TODO enable some proper tests!!”	Apache JMeter

M. A. de Freitas Farias, M. G. de Mendonça Neto, A. B. d. Silva and R. O. Spínola, “A Contextualized Vocabulary Model for identifying technical debt on code comments,” 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), Bremen, Germany, 2015, pp. 25-32, doi: 10.1109/MTD.2015.7332621.



Impact

Comment analysis considers contextual and qualitative data that can complement **quantitative** (based on metrics) and **formal** (based on parsing) analysis executed during automatic technical debt identification.

Impact

Impact on Software Quality

S Wehaibi, E Shihab, L Guerrouj examine the relation between self-admitted technical debt and software quality by investigating whether:

- Files with self-admitted technical debt have more defects compared to files without self-admitted technical debt;
- Self-admitted technical debt changes introduce future defects;
- Self-admitted technical debt-related changes tend to be more difficult.

S. Wehaibi, E. Shihab and L. Guerrouj, "Examining the Impact of Self-Admitted Technical Debt on Software Quality," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 2016, pp. 179-188, doi: 10.1109/SANER.2016.72.

Impact

Impact on Software Quality

And the results demonstrate that:

- There is no clear trend when it comes to defects and self-admitted technical debt, although the defectiveness of the technical debt files increases after the introduction of technical debt;
- Self-admitted technical debt changes induce less future defects than none technical debt changes;
- Self-admitted technical debt changes are more difficult to perform, i.e., they are more complex.

S. Wehaibi, E. Shihab and L. Guerrouj, "Examining the Impact of Self-Admitted Technical Debt on Software Quality," 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 2016, pp. 179-188, doi: 10.1109/SANER.2016.72.

Impact

Impact on Evolution

Wehaibi et al. (2019) investigated the relation between SATD and the quality of software by looking at defects. To find defects, the change history of every subject was mined to find patterns that indicate defects, such as: “**defect**”, “**bug ID”**, “**fixed issue #ID**”. The study investigated:

- The amount of defects in files with and without SATD;
- The percentage of SATD related changes that are defect-inducing;
- If changes that involve SATD files are more difficult than the ones that do not.

Impact

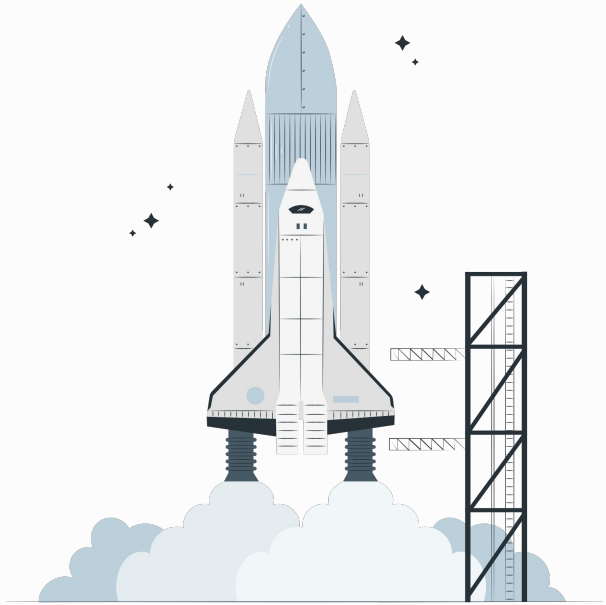
Impact on Removal

Maldonado et al. (**2017**) studied precisely this, investing how much SATD is removed from source code; who removes it; how long does it remain in a system; and what leads to removal activities.

Zampetti et al. (**2018**) conducted an in-depth quantitative and qualitative empirical study. The authors investigated how much debt was removed by accident, i.e., without the intention of resolving debt, but as a collateral of software evolution.

Zampetti, Fiorella, Alexander Serebrenik, and Massimiliano Di Penta. "Was self-admitted technical debt removal a real removal? an in-depth perspective." Proceedings of the 15th international conference on mining software repositories. 2018.

Conclusion + Future Work



- Develop tools that enable a categorized visualization of SATD to support its management;
- Develop detection approaches that inspect and analyze both, comments and source code for improved accuracy;
- Proposing new approaches and techniques to mitigate and repay debt;
- Investigate new measures to estimate the effort required to repay SATD;
- Study the presence of SATD in other software artifacts, such as the messages and descriptions of issues and commits

And much more…

Thanks!

Do you have any questions?

altino@ufmg.br