An Exploratory Study on Life Cycle of Technical Debt in Issue Trackers: from Introduction to Repayment - Codebook

YIKUN LI, University of Groningen, the Netherlands
MOHAMED SOLIMAN, University of Groningen, the Netherlands
PARIS AVGERIOU, University of Groningen, the Netherlands

DEFINITIONS

Table 1 presents the indicators for each Technical Debt type, as well as their definitions. The *Reused* column refers to whether the indicators are reused as-is from the study of Alves *et al.* ("●" symbol) or they were created by us ("○" symbol). The original definitions of the Technical Debt types and examples of indicators from the study of Alves *et al.* are shown in Table 2.

Table 1. Definitions of indicators of different types of technical debt in issue trackers.

Type	Indicator	Reused	Definition
Architecture Debt	Violation of modularity	●	Because shortcuts were taken, multiple modules became inter-dependent, while they should be independent.
	Using obsolete technology	○	Architecturally-significant technology has become obsolete.
Build Debt	Under- or over-declared dependencies	●	Under-declared dependencies: dependencies in upstream libraries are not declared and rely on dependencies in lower level libraries. Over-declared dependencies: unneeded dependencies are declared.
	Poor Deployment Practice	○	The quality of deployment is low that compile flags or build targets are not well organized.
Code Debt	Complex code	○	Code has accidental complexity and requires extra refactoring action to reduce this complexity.
	Dead code	○	Code is no longer used and needs to be removed.
	Duplicated code	●	Code that occurs more than once instead of as a single reusable function.
	Low-quality code	○	Code quality is low, for example because it is unreadable, inconsistent, or violating coding conventions.
	Multi-thread correctness	●	Thread-safe code is not correct and may potentially result in synchronization problems or efficiency problems.
	Slow algorithm	●	A non-optimal algorithm is utilized that runs slowly.
Defect Debt	Uncorrected known defects	●	Defects are found by developers but ignored or deferred to be fixed.
Design Debt	Non-optimal decisions	○	Non-optimal design decisions are adopted.
Documentation Debt	Outdated documentation	●	A function or class is added, removed, or modified in the system, but the documentation has not been updated to reflect the change.
	Low-quality documentation	○	The documentation has been updated reflecting the changes in the system, but quality of updated documentation is low.
Requirements Debt	Requirement partially implemented	○	Requirements are implemented, but some are not fully implemented.
	Non-functional requirements not fully satisfied	○	Non-functional requirements (e.g. availability, capacity, concurrency, extensibility), as described by scenarios, are not fully satisfied.
Test Debt	Expensive tests	○	Tests are expensive, resulting in slowing down testing activities. Extra refactoring actions are needed to simplify tests.
	Lack of tests	○	A function is added, but no tests are added to cover the new function.
	Low coverage	●	Only part of the source code is executed during testing.

Table 2. Original indicators by type of technical debt by Alves *et al.* [1]

Type	Definition	Indicator
Architecture Debt	Refers to the problems encountered in project architecture, for example, violation of modularity, which can affect architectural requirements (performance, robustness, among others). Normally this type of debt cannot be paid with simple interventions in the code, implying in more extensive development activities.	ACN/PWDR Betweeness centrality Issues in software architecture Structural analysis Structural dependencies Violation of modularity
Build Debt	Refers to build related issues that make this task harder, and more time/processing consuming unnecessarily. The build process of a project can contain very unnecessary code to the customer. Moreover, if the build process needs to run ill-defined dependencies, the process becomes unnecessarily slow. When this occurs, one can identify a build debt.	Dead flags Zombie targets Dependency Visibility
Code Debt	Refers to the problems found in the source code which can affect negatively the legibility of the code making it more difficult to be maintained. Usually, this debt can be identified by examining the source code of the project considering issues related to bad coding practices.	ASA issues Code metrics Code outside of standards Duplicated code Multi-thread correctness (ASA) Slow algorithm
Defect Debt	Software projects may have known and unknown defects in the source code. Defect debt consists of known defects, usually identified by testing activities or by the user and reported on bug track systems, that the CCB agrees should be fixed, but due to competing priorities, and limited resources have to be deferred to a later time. Decisions made by the CCB to defer addressing defects can accumulate a significant amount of technical debt for a product making it harder to fix them later.	Uncorrected known defects
Design Debt	Refers to debt that can be discovered by analyzing the source code by identifying the use of practices which violated the principles of good object-oriented design (e.g. very large or tightly coupled classes).	ASA issues Brain method Code metrics Code smells Data class Data clumps Dispersed coupling duplicated code God class (or large class) Grime Intensive coupling Issues in the software design Refused parent bequest Schizophrenic class Structral analysis
Documentation Debt	Refers to the problems found in software project documentation and can be identified by looking for missing, inadequate, or incomplete documentation of any type. Inadequate documentation is those that currently work correctly in the system, but fail to meet certain quality criteria of software projects.	Documentation does not exist Incomplete design specification Incomplete documentation Insufficient comments in code Outdated documentation Test documentation
Infrastructure Debt	Refers to infrastructure issues that, if present in the software organization, can delay or hinder some development activities. Some examples of this kind of debt are delaying an upgrade or infrastructure fix.	-
People Debt	Refers to people issues that, if present in the software organization, can delay or hinder some development activities. An example of this kind of debt is expertise concentrated in too few people, as an effect of delayed training and/or hiring.	-
Process Debt	Refers to inefficient processes, e.g. what the process was designed to handle may be no longer appropriate.	-
Requirements Debt	Requirements debt refers to tradeoffs made with respect to what requirements the development team need to implement or how to implement them. Some examples of this type of debt are: requirements that are only partially implemented, requirements that are implemented but not for all cases, requirements that are implemented but in a way that doesn't fully satisfy all the non-functional requirements (e.g. security, performance, etc.).	Requirement backlog list
Service Debt	The need for web service substitution could be driven by business or technical objectives. The substitution can introduce a TD, which needs to be managed, cleared and transformed from liability to value-added. Technical debt can cover several dimensions, which are related to selection, composition, and operation of the service.	Selection/replacement of web service
Test Automation Debt	Test Automation debt is defined as the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles.	-
Test Debt	Refers to issues found in testing activities which can affect the quality of testing activities. Examples of this type of debt are planned tests that were not run, or known deficiencies in the test suite (e.g. low code coverage).	Incomplete tests Low coverage

REFERENCES

[1] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spínola. 2014. Towards an ontology of terms on technical debt. In *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 1–7.