Towards an Ontology of Terms on Technical Debt

Nicolli S. R. Alves[1], Leilane F. Ribeiro[1], Vivyane Caires[1], Thiago S. Mendes[2,3], Rodrigo O. Spínola[1,2]
[1]Graduate Program in Systems and Computer
Salvador University – UNIFACS, Salvador, Brazil
[2] Fraunhofer Project Center for Software and System Engineering
Federal University of Bahia – UFBA, Salvador, Brazil
[3]Information Technology Department,
Federal Institute of Bahia – IFBA, Santo Amaro, Bahia, Brazil
nicollirioss@gmail.com, leilanefr@yahoo.com.br, vivyane.caires@gmail.com,
thiagomendes@dcc.ufba.br, rodrigo.spinola@pro.unifacs.br

Abstract—**Technical debt is a term that has been used to describe the increased cost of changing or maintaining a system due to shortcuts taken during its development. As technical debt is a recent research area, its different types and their indicators are not organized yet. Therefore, this paper proposes an ontology of terms on technical debt in order to organize a common vocabulary for the area. The organized concepts derived from the results of a systematic literature mapping. The proposed ontology was evaluated in two steps. In the first one, some ontology design quality criteria were used. For the second one, a specialist in the area performed an initial evaluation. This work contributes to evolve the Technical Debt Landscape through the organization of the different types of technical debt and their indicators. We consider this an important contribution for both researchers and practitioners because this information was spread out in the literature hindering their use in research and development activities.**

Keywords—technical debt; technical debt types; technical debt indicators; ontology; software maintenance

I. Introduction

The Technical Debt (TD) metaphor was first mentioned by Ward Cunningham in 1992 and was described as those internal tasks you choose do not to perform now, but that run the risk of causing future problems if not done [1]. Thus, the metaphor defines the debt that the development team takes when it opts for an easy approach to implement in the short term, but that has a great possibility of having a negative impact in the long term.

Lately, the term TD has been used to describe different issues during the software development life cycle, covering aspects that impact negatively on deployment activities, evolution of a system or any obstacle that hinders the progress of activities involved in its development [2].

The acceptance and use of the TD metaphor is in large part because it is easily understood [36]. However, as TD is a new research area, having published papers only from 2010, its different types and indicators are not organized yet and are currently spread out through a lot of papers making it difficult to define a common vocabulary for the area and to indicate which directions to follow in order to find out the existing debt on software projects.

There are some initiatives in order to organize the different types of TD. For example, Martin Fowler [6] classified the debts considering the following characteristics: Reckless/Prudent and Deliberate/Inadvertent. These characteristics compose what he called Technical Debt Quadrant and allow to classify the debt analyzing if it was inserted intentionally or not and, in both cases, if it can be considered the result of a careless action or was inserted with prudence. Not so far from the Fowler's classification, Steve McConnell [7] organized the debt into two groups: intentional and unintentional. In another related work, the types of debt were organized in dimensions [37]. To organize these dimensions, the authors performed a multivocal literature review and a set of interviews with software practitioners and academics. As a result, a list of seven dimensions of debt was initially identified.

However, those initiatives do not consider the nature of the debt as a factor to be considered in its classification. By nature we mean the activity of the development process execution where the debt was inserted or it is associated with. For example, a debt incurred by a tester who does not execute a set of planned test scenarios can be considered a test debt. Another example is a developer to take an inadequate design decision to solve a short term problem at the expense of a more robust solution; this decision may cause the insertion of a design or architecture debt.

In this context, this paper proposes an ontology to organize the different types of technical debt considering their nature as a classification criterion. Based on this organization, it was possible to identify different indicators that have been proposed to find out each type of TD on software projects. It is important to mention that all organized knowledge was originated from the results of a systematic literature mapping whose results will be available soon.

The proposed ontology of terms in technical debt was assessed in two steps. In the first one, the quality criteria defined in [13] were considered. In the second step, an expert that did not participate of the ontology development performed an initial evaluation of the organized knowledge.

The first contribution of this work is the organization of the different types of TD. This organization contributes to the evolution of the Technical Debt Landscape [22] [2]. Besides, as second contribution, the indicators that have been used or proposed to support the identification of TD items on software projects were organized too. The identified TD types and their respective indicators were structured in an ontology, which allows sharing of a common vocabulary for the research community in TD and practitioners.

Besides this introduction, this paper has other three sections. In section II, the definitions of ontology, its types and the process used to create the proposed ontology will be presented. Then, in section III, the ontology for the technical debt area will be discussed. Finally, section IV presents some final remarks and future works.

II. Background

Knowledge representation and organization systems are considered fundamental processes amid rising information production [3]. Taxonomies and ontologies are tools that have been used as the basis for the development of these types of systems. Taxonomies allow to organize information and/or knowledge in hierarchical relationships between terms [8]. On the other side, ontology is a vocabulary representation that is often specialized to some domain or subject [4]. Thus, it supports the capture, representation, search, storage and standardization of knowledge, describing a consistent, complete and unambiguous vocabulary [5].

Different types of ontologies can be developed according to their level of generality. They can be classified as Top-level, Domain, Task, and Application Ontology [5]. According to Guarino [5], a top-level ontology describes generic elements that are not part of a specific domain such as space, time, action, event and objects. A domain ontology maps concepts of a particular domain, specializing the terms of a top-level ontology. The domain ontology must specify the relationships, rules and exceptions of all objects present in the domain conceptualization. Although the task ontology is similar to domain ontology, it differs by describing a vocabulary related to a task or activity through specialization of the concepts introduced in a top-level ontology [10]. And finally, an application ontology contains concepts that belong simultaneously to a domain and a task, through specialization of the concepts of a domain and a task ontology [5].

It is still possible to classify ontologies according to the detail level of their internal structure as lightweight and heavyweight [10]. The first one considers basically the concepts, their properties and relationships between them. On the other side, heavyweight ontologies define the knowledge more deeply and add axioms and constraints in order to clarify the meaning of terms [10]. In this paper, it was developed a **lightweight domain ontology**.

The development of domain ontologies is not a simple task. Like any other activity of conceptual modeling, this activity should be supported by software engineering practices. A systematic approach for constructing ontologies (SABiO) was defined in [11]. Although there are other approaches, SABiO was chosen in this work because it has already been used to support the definition of different ontologies [14]. SABiO considers the following activities [11]:

- **Purpose Identification and Requirements Specification:** clearly identify the purpose and intended use of the ontology, i.e., the competence of the ontology;
- **Ontology Capture:** the most important step in the development of the ontology. Its goal is to capture the domain's concept based on the ontology competency. In addition, a model using a graphical language with a dictionary of terms must be used to facilitate the communication with domain specialists;
- **Ontology Formalization:** a formalism to represent the ontology must be used;
- **Integrating Existing Ontologies:** during capture and/or formalization processes, it may be necessary to integrate the current ontology with existing ones in order to take advantage of concepts already established previously [12];
- **Ontology Evaluation:** the ontology should be evaluated to determine whether the specification satisfies its requirements. Moreover, it can also be evaluated against some design quality criteria [13];
- **Ontology Documentation:** all ontology development must be documented, including purposes, requirements and motivating scenarios, textual descriptions of conceptualization, the formal ontology and the adopted design criteria.

This process must be seen as an iterative process rather than sequential steps. For example, the ontology capture step can point to new requirements or, during the evaluation; one can see that the identified terms are not sufficient to conclude the ontology.

III. Ontology of Terms on Technical Debt

A. Purpose Identification and Requirements Specification

This is the first activity when building an ontology. In this activity the competence of the ontology is identified, i.e., its use and purpose, through the delimitation of what is or not relevant for it. In this work, the purpose of the ontology is to organize the different TD types considering its nature as classification criterion. Thus, the following competency question was defined:

What are the types of technical debt and their indicators that can be considered in software projects?

The answer to this question is important because until now, to the best of our knowledge, there is no initiative in order to organize the types of TD considering their nature as a classification perspective as well as their current indicators.

B. Ontology Capture and Formalization

Ontology capture involves the identification and specification of concepts (classes), their relationships and all other elements necessary for the representation of the ontology, such as properties, axioms, instances, etc.. As indicated previously, in this study the relevant elements will be identified to define a lightweight ontology.

The types of TD as well as their definitions were identified from the results of a systematic literature mapping performed in the area. Table I shows the definitions of each identified TD type. Those types do not intend to be mutually exclusive and their relationship with each other will be discussed later on this paper.

Associated with each TD type, other relevant information's were also specified: indicators that can be used to identify the TD in software projects, and references where the information was extracted. Table II shows the indicators that were identified in the technical literature for each TD type.

Analyzing Table II, it is possible to observe that, although indicators have not yet been found or proposed for some types of TD (process, infrastructure, test automation, and people), the number of indicators is already relatively large. However, few of them were effectively evaluated through experimental studies.

After capturing the ontology concepts, their formalization was started. Several languages can be used for this purpose, such as OWL (Web Ontology Language) [15] [16], SHOE (Simple HTML Ontology Extensions) [17], XOL (Ontology Exchange Language) [19], OIL (Ontology Inference Layer) [18] and DAML (DARPA Agent Markup Language) [20]. In this work, OWL was used.

Among the reasons considered for choosing OWL, it is considered a standard for the development of ontologies (recommended by W3C), because it was designed to be used by applications that need to process the content of information instead of just presenting it. Fig. 1 shows a fragment of the ontology on TD defined in OWL. The fragment represents the concept of design debt. Note that the TD types were defined as subclasses of Technical Debt and other information (definition, indicators and references) were specified using *Annotations.*

TABLE I. IDENTIFIED TECHNICAL DEBT TYPES DEFINITIONS

Type	Definition
Architecture Debt [2] [26] [37]	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.
Build Debt [27]	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.
Code Debt [28] [35] [37]	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.
Defect Debt [29]	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.
Design Debt [22] [30] [35] [37]	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).
Documentation Debt [30] [37]	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.
Infrastructure Debt [34]	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 [34] [37]	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 [31]	Refers to inefficient processes, e.g. what the process was designed to handle may be no longer appropriate.
Requirement Debt [2]	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.).
Service Debt [33]	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.
Test Automation Debt [32]	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 [30] [37]	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).

Besides, disjoint classes restriction was also used. Classes are assumed to overlap. Thus, we cannot assume that an individual is not a member of a particular class simply because it has not been asserted to be a member of that class. In order to separate a group of classes we must make them disjoint from one another. This ensures that an individual who has been asserted to be a member of one of the classes in the group cannot be a member of any other classes in that group. For example, a TD item cannot be categorized at the same time as process and test debt because it does not make sense. Thus, test and process debt are considered disjoint classes. On the other side, it is also possible that, in some cases, TD items are categorized in more than one type. For example, a problem at the design level can be categorized as a design or architecture debt. For situations like this, the disjunction constraint was not defined. Table III shows the disjoint classes for each identified TD type. The marked cells indicate that there is a disjunction constraint between TD types.

TABLE II. INDICATORS BY TYPE OF TECHNICAL DEBT

TD Type	Indicator
Architecture Debt	ACN/PWDR Betweeness Centrality Issues in software architecture Structural Analysis Structural Dependencies Violation of Modularity
Build Debt	“Dead Flags” “Zombie Targets” Dependency Visibility
Code Debt	ASA Issues Code Metrics Code outside of standards Duplicated code Multithread correctness (ASA) Slow Algorithm
Defect Debt	Uncorrected known defects
Design Debt	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 Structural Analysis
Documentation Debt	Documentation does not exist Incomplete Design Specification Incomplete Documentation Insufficient comments in code Outdated Documentation Test Documentation
Infrastructure Debt	-
People Debt	-
Process Debt	-
Requirement Debt	Requirement Backlog List
Service Debt	Selection/Replacement of web service
Test Automation Debt	-
Test Debt	Incomplete Tests Low coverage

Protégé [21] tool was used to support the formalization of the ontology in OWL. Fig. 2 shows the ontology visual representation.

C. Ontology Evaluation

The ontology of terms on TD was evaluated considering its requirements and some design quality criteria. This process was performed in two steps: (1) evaluation regarding the quality criteria defined in [13]; (2) evaluation by an expert in the area.

```
<?xml version="1.0"?>
...

<Ontology xmlns="http://www.w3.org/2002/07/owl#"
...
    <Declaration>
        <Class IRI="#Design_Debt"/>
    </Declaration>
    <SubClassOf>
        <Class IRI="#Design_Debt"/>
        <Class IRI="#Technical_Debt"/>
    </SubClassOf>
    <DisjointClasses>
        <Class IRI="#Automation_debt"/>
        <Class IRI="#Build_Debt"/>
        <Class IRI="#Defect_Debt"/>
        <Class IRI="#Documentation_Debt"/>
        <Class IRI="#Infrastructure_Debt"/>
        <Class IRI="#People_Debt"/>
        <Class IRI="#Process_Debt"/>
        <Class IRI="#Requirement_Debt"/>
        <Class IRI="#Service_Debt"/>
        <Class IRI="#Test_Debt"/>
    </DisjointClasses>
    <AnnotationAssertion>
        <AnnotationProperty IRI="#definition"/>
        <IRI>#Design_Debt</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">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 (eg very large or tightly coupled
classes).</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty IRI="#indicator"/>
        <IRI>#Design_Debt</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">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, Structural Analysis
</Literal>
    </AnnotationAssertion>
    <AnnotationAssertion>
        <AnnotationProperty abbreviatedIRI="rdfs:isDefinedBy"/>
        <IRI>#Design_Debt</IRI>
        <Literal datatypeIRI="&rdf;PlainLiteral">N. Zazworka, M.
A. Shaw, F. Shull, and C. Seaman, â€œInvestigating the impact of
design debt on software quality,â€  in Proceeding of the 2nd
working on Managing technical debt, ser. MTD â€™11. New York, NY,
USA: ACM, 2011, pp. 17â€"23.</Literal>
    </AnnotationAssertion>
...
</Ontology>
```

Fig. 1. Fragment of the ontology defined in OWL.

TABLE III. DISJUNTION RESTRICTION BETWEEN TD TYPES

	Architecture Debt	Build Debt	Code Debt	Defect Debt	Design Debt	Documentation Debt	Process Debt	Infrastructure Debt	Test Automation Debt	People Debt	Requirement Debt	Service Debt	Test Debt
Architecture Debt		X	X	X		X	X	X	X	X	X	X	X
Build Debt	X			X	X	X	X	X	X	X	X	X	X
Code Debt	X			X			X	X	X	X	X	X	X
Defect Debt	X	X	X		X	X	X	X	X	X	X	X	X
Design Debt		X		X		X	X	X	X	X	X	X	X
Documentation Debt	X	X		X	X		X	X	X	X	X	X	X
Process Debt	X	X	X	X	X	X		X	X	X	X	X	X
Infrastructure Debt	X	X	X	X	X	X	X		X	X	X	X	X
Test Automation Debt	X	X	X	X	X	X	X	X		X	X	X	
People Debt	X	X	X	X	X	X	X	X	X		X	X	X
Requirement Debt	X	X	X	X	X	X	X	X	X	X		X	X
Service Debt	X	X	X	X	X	X	X	X	X	X	X		X
Test Debt	X	X	X	X	X	X	X	X		X	X	X	

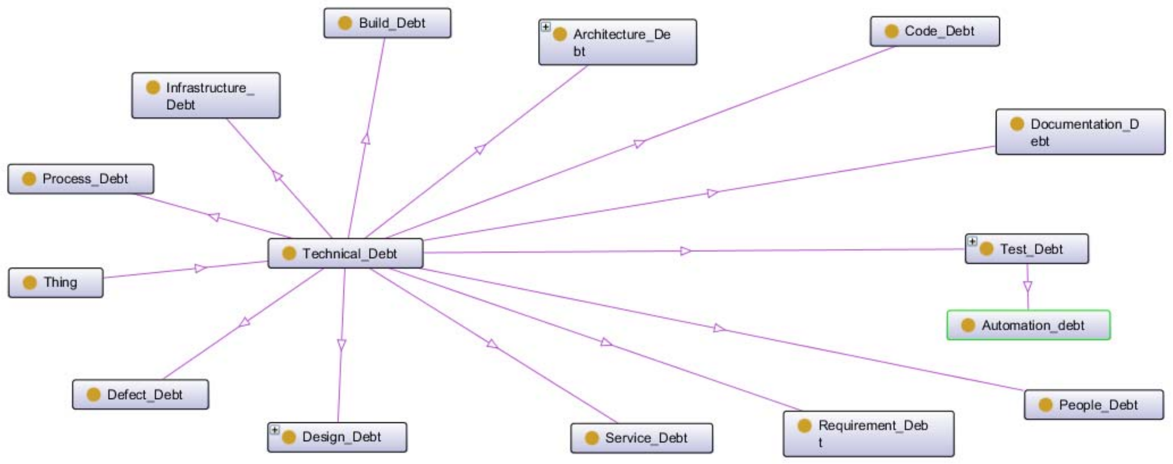

Fig. 2. Ontology visual representation in Protegé.

For the first step, the ontology was sent to two of the researchers involved in the writing of this paper, but different from the one responsible for the ontology definition. For each TD type, the researcher was asked to evaluate it considering the following possible results:

- **Fully Compliance (FC):** the ontology complies with all variables considered in the criterion being evaluated;
- **Partially Compliance (PC):** the ontology is partially consistent with the variables considered in the criterion being evaluated;
- **Not Compliance (NC):** the ontology does not comply with all variables considered in the criterion.

If the evaluation result was Partially Compliance or Not Compliance, the evaluators should report the identified issues so that the ontology could be improved.

The result of this evaluation can be seen in Table IV. It shows the used quality criteria [13], and the result of their application. In general, minor adjustments were required in order to clarify the differences between some TD types and also to improve the explanation of some examples. From these results, improvements were made in the ontology so that it could address the reported non-compliances.

The second step of the evaluation considered the recommendations of [5] [13] that indicate that ontologies should be based on the consensus of a group of specialists. In order to get an initial feedback, the defined ontology on TD was evaluated by a specialist in the area who was not involved on its definition. For this, the identified types and their definitions were organized in a form that was sent for review.

As result, some improvements were suggested in order to clarify some definitions that, in some cases, were described from different points of view ("*this partially comes from the fact that you have different types of definitions. Some of your definitions describe a type of debt from the point of view of how the debt is identified, while other definitions are more about how the debt was incurred*"). Furthermore, it was also indicated that the identified types make sense, and no new type was suggested ("*I think the descriptions are very clear...There are some types of debt you have here that I hadn't thought of before, but they make sense. I can't think of any other types that you haven't included.*"). Finally, the expert mentioned that there is still no consensus in the TD research community if the requirements debt can actually be considered as a type of technical debt ("*One other*

TABLE IV. ONTOLOGY EVALUATION REGARDING QUALITY CRITERIA DEFINIED BY [13].

Criteria	Evaluation		Remarks
	1	2	
Clarity: An ontology should effectively communicate the intended meaning of defined terms. Definitions should be objective and, when a definition can be stated in logical axioms, it should be. All definitions should be documented with natural language [13].	PC	PC	Logical axioms were not used in the formalization of the ontology because it is a lightweight ontology and the formalized knowledge does not require their use. On the other side, the TD types were defined and documented. Some minor adjustments were required in order to clarify the difference between some TD types.
Coherence: an ontology should sanction inferences that are consistent with the definitions. At the least, the defining axioms should be logically consistent. Coherence should also apply to the concepts that are defined informally [13].	FC	FC	The defined ontology did not use axioms. Thus, this criterion was used to evaluate only the documented definitions. It was not identified any issue for this criterion.
Extendibility: An ontology should be designed to anticipate the uses of the shared vocabulary. In other words, one should be able to define new terms form special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions [13].	FC	PC	The defined ontology is the first step towards a more comprehensive knowledge organization in TD considering its types, causes and indicators. Thus, it is expected that it will be evolved from the formalized definitions. In this context, one of evaluators requested that some definitions were improved to facilitate their extension.
Minimal encoding bias: The conceptualization should be specified at the knowledge level without depending on a particular symbol-level encoding [13].	FC	FC	The defined and documented concepts in the ontology were not influenced by the restrictions of the chosen language (OWL) for their representation.
Minimal ontological commitment: An ontology should require the minimal ontological commitment sufficient to support the intended knowledge sharing activities [13].	FC	FC	The defined ontology organizes a common vocabulary for the DT area, is extensible and does not use a very extensive formalism making its use easier to the research community.

issue is requirements debt. There is a lot of disagreement in the TD research community about whether this should be considered technical debt or not”). Nevertheless, considered it important to keep the concepts in the proposed ontology because this can be a good material for discussion by the research community.

The improvement suggestions were made in the ontology whose definitions are already adjusted in Table I.

IV. CONCLUSION

Technical debt is a recent research area and brings a series of challenges and opportunities. In this paper it was proposed an organization of the types of TD considering its nature as a classification criterion.

This work contributes to the Technical Debt Landscape [22] [2] through the organization of the different types of technical debt that have been considered in the technical literature. The identified types were organized using an ontology, which will allow the sharing of a common vocabulary for the research community on TD.

Furthermore, indicators that have been used or proposed to support the identification of TD items on software projects were also organized for each identified type. We consider this an important contribution of this work for researchers and practitioners. The indicators were scattered in the literature hindering their use in TD items identification activities/research.

This work is in the context of a more comprehensive research project with the purpose of mapping, beyond the TD types and their indicators, the main causes of the occurrence of each TD type. The defined ontology will be the basis for structuring this knowledge.

The definition of an ontology is a big challenge, especially when the ontology is expected to have relevance and value to a broad audience [24]. As the definition of the TD types is an initial attempt to organize the knowledge on TD, the list of types presented in this work may evolve to the exclusion or inclusion of some of them. In order to obtain a better defined domain ontology, the collaboration between different experts in the area needs to be stimulated allowing each of them to participate actively in the development of the ontology [23] [25]. So, as the next steps of this research, the authors are considering to evaluate the ontology in a more comprehensive way. To support this task, we are working on a web based infrastructure (wrapping the defined ontology) to allow the sharing and the collaborative maintenance and evolvement of this knowledge on TD.

Besides that, our research group is also working on a set of tools to support the visualization of TD on software projects. The choice of which indicators could be useful to identify TD items and which visual metaphors could be used to represent them will be supported by the ontology proposed on this work.

Acknowledgment

The authors would like to thank Dr. Carolyn Seaman for her comments and insights that helped to improve the ontology of terms on Technical Debt. The authors also would like to thank student Rodrigo Silva that is helping with some technical stuff, and CAPES for the financial support to this work.

References

[1] W. Cunningham, "The Wycash Portfolio Management System," in ACM SIGPLAN OOPS Messenger (Vol. 4, No. 2). ACM. December 1992, pp. 29-30.

[2] P. Kruchten, R. L. Nord, I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," IEEE Software, Published by the IEEE Computer Society, November 2012.

[3] J. Fogl, "Relations of the Concepts 'Information' and 'Knowledge'," International Fórum on Information and Documentation, v. 4, n. 1, 1979, p. 21-24.

[4] B. Chandrasekaran, J. R. Josephson, V. R. Benjamins, "What Are Ontologies, and Why Do We Need Them?," IEEE Intelligent Systems, January/February 1999.

[5] N. Guarino, "Formal Ontology and Information Systems," in Proceedings of International Conference in Formal Ontology and Information Systems – FOIS'98, Trento, Italy, 1998.

[6] M. Fowler, "Technical Debt Quadrant," Bliki [Blog]. Available from: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html, 2009.

[7] S. McConnell, "Technical Debt. 10x Software Development," [Blog]. Available at: http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx, 2007.

[8] K. C. Adams, "Immersed In Structure: The Meaning An Function Of Taxonomies". Available at: http://www.internettg.org/newsletter/avg00/contents.html, 2000.

[9] R. Van Rees, "Clarity In The Usage Of The Terms Ontology, Taxonomy And Classification". Available at: http://reinout.vanrees.org/_downloads/2003_cib.pdf, 2003.

[10] C. Calero, F. Ruiz, M. Piattini (eds), "Ontologies for software engineering and software technology," Springer-Verlag, Germany, 2006.

[11] R. A. Falbo, C.S. Menezes, A.R.C. Rocha, "A Systematic Approach for Building Ontologies," Proc. of the 6th Ibero-American Conference on Artificial Intelligence, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.

[12] M. Uschold, M. King, "Towards a Methodology for Building Ontologies," Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI', 1995.

[13] T.R. Gruber, "Toward Principles For The Design Of Ontologies Used For Knowledge Sharing," Int. Journal Human-Computer Studies, 43(5/6), p. 907-928, 1995.

[14] R. A. Falbo, "Experiences in Using a Method for Building Domain Ontologies," Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering, SEKE'2004, pp. 474-477, International Workshop on Ontology In Action, OIA'2004. Banff, Alberta, Canada, 2004.

[15] W3C, "OWL 2 Web Ontology Language – Document Overview," W3C Recommendation. Available at: http://www.w3.org/TR/owl2-overview/.

[16] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," W3C Recommendation. Available at: http://www.w3.org/TR/owl-features/, 2004.

[17] SHOE. Home Page. http://www.cs.umd.edu/projects/plus/SHOE/.

[18] OIL. Home Page. http://www.ontoknowledge.org/oil/.

[19] XOL. Home Page. http://www.ai.sri.com/pkarp/sol.

[20] DAML. Home Page. http://www.daml.org/.

[21] Protegé, Software Protegé. Available at: http://protege.stanford.edu/, 2011.

[22] C. Izurieta, A. Vetró, N. Zazworka, Y. Cai, C. Seaman, F. Shull, "Organizing the Technical Debt Landscape," IEEE ACM MTD 2012 3rd International Workshop on Managing Technical Debt. In association with the 34th International Conference on Software Engineering ICSE, Zurich, Switzerland, 2012.

[23] M. Rospocher, C. Ghidini, C. Di Francescomarino, "Evaluating Wiki Collaborative Features in Ontology Authoring," IEEE Transactions on Knowledge and Data Engineering, IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2312325, 2014.

[24] Y. Chen, S. Zhang, X. Peng, W. Zhao, "A Collaborative Ontology Construction Tool with Conflicts Detection," Knowledge and Grid, 2008. SKG'08. Fourth International Conference on Semantics, 3-5, IEEE Computer Society Digital Library, 2008.

[25] B. Jie, H. Zhiliang, C. Doina, R. James, H. Vasant G, "A Tool for Collaborative Construction of Large Biological Ontologies," Proceedings of the 17th International Conference on Database and Expert Systems Applications; 2006. pp. 191–5

[26] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems", FoSER '10: Proceedings of the FSE/SDP workshop on Future of software engineering research, 2010.

[27] J. Morgenthaler, M. Gridnev, R. Sauciuc and S. Bhansali, "Searching for build debt: Experiences managing technical debt at Google," in Third International Workshop on Managing Technical Debt, 2012, pp. 1-6.

[28] J. Bohnet, and J. Dџllner, "Monitoring code quality and development activity by software maps", MTD '11: Proceedings of the 2nd Workshop on Managing Technical Debt, 2011.

[29] W. Snipes, B. Robinson, Y. Guo and C. Seaman, "Defining the decision factors for managing defects: A technical debt perspective", in Third International Workshop on Managing Technical Debt (MTD), 2012, pp. 54-60.

[30] C. Seaman and Y. Guo, "Measuring and Monitoring Technical Debt,", Advances in Computers 82, 2011, pp. 25-46,.

[31] Z. Codabux, and B. Williams, "Managing technical debt: An industrial case study," in 4th International Workshop on Managing Technical Debt (MTD), 2013, pp. 8-15.

[32] K. Wiklund, S. Eldh, D. Sundmark and Lundqvist, K. "Technical Debt in Test Automation," in IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012, pp. 887-892.

[33] E. Alzaghoul and R. Bahsoon, "CloudMTD: Using real options to manage technical debt in cloud-based service selection, in 4th International Workshop on Managing Technical Debt (MTD), 2013, pp. 55-62.

[34] C. Seaman and R. O. Spínola, "Managing Technical Debt," [Short Course] XVII Brazilian Symposium on Software Quality, Salvador, Brazil, 2013.

[35] N. Zazworka, R. O. Spínola, A. Vetró, F. Shull and C. Seaman, "A case study on effectively identifying technical debt," EASE '13: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, 2013.

[36] R. O. Spínola, N. Zazworka, A. Vetró, C. Seaman and F. Shull, "Investigating technical debt folklore: Shedding some light on technical debt opinion," in 4th International Workshop on Managing Technical Debt (MTD), 2013, pp. 1-7.

[37] E. Tom, A. Aurum, and R. Vidgen. "An exploration of technical debt". Journal of Systems and Software. 86, 6 (June 2013), 1498-1516. DOI=10.1016/j.jss.2012.12.052.