View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by AIS Electronic Library (AISeL)

A Collaborative Computational Infrastructure for Supporting Technical Debt Knowledge Sharing and Evolution

Full Paper

Nicolli S. R. Alves
Salvador University, Salvador, Brazil
nicollirioss@gmail.com

Rodrigo S. de Araújo
Salvador University, Salvador, Brazil
rod.dearaujo@gmail.com

Rodrigo O. Spínola
Salvador University, Salvador, Brazil
Fraunhofer Project Center for Software and System Eng. at UFBA, Salvador, Brazil
rodrigo.spinola@pro.unifacs.br

Abstract

Keeping information systems useful during their evolution is a complex task. This complexity usually comes from the lack of concern with their maintainability. The monitoring of technical debt (TD) is one way to minimize the effects of low maintainability. But even before developers can monitor the debt, they need to understand what TD types can be incurred, how they can be identified, and which causes can lead developers to incur them into the project. Nevertheless, despite the importance of to know what TD is, this knowledge is still spread out hindering a common understanding of the area. In this context, this paper presents TD Wiki, a collaborative computational infrastructure for supporting TD knowledge sharing and evolution through the using of knowledge visualization techniques. TD Wiki is already available for using and provides useful information that can be used when developing strategies for TD monitoring during the development of information systems.

Keywords

Technical debt, Knowledge sharing and evolution; TD Wiki; Software maintenance and evolution.

Introduction

Rather than developing new information systems, keeping it useful during its evolution so that it follows the organizational changes is a complex task. The difficulties involved in this task are often associated to the fact that information systems have been developed without a major concern with their maintainability. The maintainability is a software quality characteristic that indicates the facility that the software can be understood, corrected, adapted, and/or improved (Pfleeger and Atlee 2009). Several factors can positively or negatively impact the presence of this characteristic in the project such as using of good object oriented design principles, design patterns, comments in the code, and updated design documentation. The monitoring of technical debt (TD) keeping it within levels where it can be administered is one way to minimize the effects of low maintainability in software projects.

TD represents the effects of immature artifacts in software maintenance that brings a short-term benefit in terms of increased productivity and lower costs, but may have to be adjusted later with interest (Seaman and Guo 2011; Kruchten et al. 2012; Izurieta et al. 2012). TD is usually incurred when you need to choose between keeping the system under quality standards or putting it to work in the shortest possible time, using minimal resources. The principal on the debt refers to the work left to be done. The interest is the potential penalty in terms of increased effort and decreased productivity that will have to be

paid in the future as a result of not completing these tasks in the present, including the extra cost of paying off the debt later, as compared to earlier (Seaman and Guo 2011).

But even before the development team can work on the monitoring of debt, it needs to understand what types of debt can be incurred, how they can be identified, and which causes can lead the development team to incur them into the project. Nevertheless, although technical debt is being increasingly discussed, as reported by trends.google.com (see Figure 1 indicating that over the last seven years more and more Google users have been searching for the term "Technical Debt"), knowledge on TD is still spread out in the technical literature hindering a common understanding of the area. Alves et *al.* (2014) proposed an organization of different types of debt considering its nature as classification criteria. This organization resulted in an ontology of terms on TD. On its first version, the ontology mapped the types of debt and their indicators. We evolved this ontology by adding some causes that lead the development team to incur each type of debt.

Despite ontology be a powerful mechanism to represent the knowledge, it is difficult for people who are not specialist on knowledge representation mechanisms to understand and manipulate it. This concern is still more critical when the knowledge needs to be shared with a medium to large audience as we see on information systems development teams. Thus, the sharing and evolution of this knowledge need to be stimulated in a way that both researchers and practitioners have access to it and can use it in their daily activities.

In this context, this paper presents TD Wiki, a computational infrastructure for supporting the sharing and collaborative evolution of knowledge on TD through the using of knowledge visualization techniques. There are two main expectations for this infrastructure: (1) to promote the effective dissemination of TD knowledge in order to stimulate its effective using by practitioners; (2) to allow the collaborative evolution of the organized knowledge. Thus, by providing necessary information for development teams start to monitor TD on their projects, TD Wiki can positively contribute to improve the quality of information systems.

In addition to this introduction, this paper has more four sections. In the next section we will present some definitions used in the development of this work. After, the organized knowledge on TD will be discussed. Based on this knowledge, TD Wiki will be presented on the next section considering its requirements, architecture, and the implemented computational infrastructure itself. Finally, the final remarks and next steps of this research will be discussed.

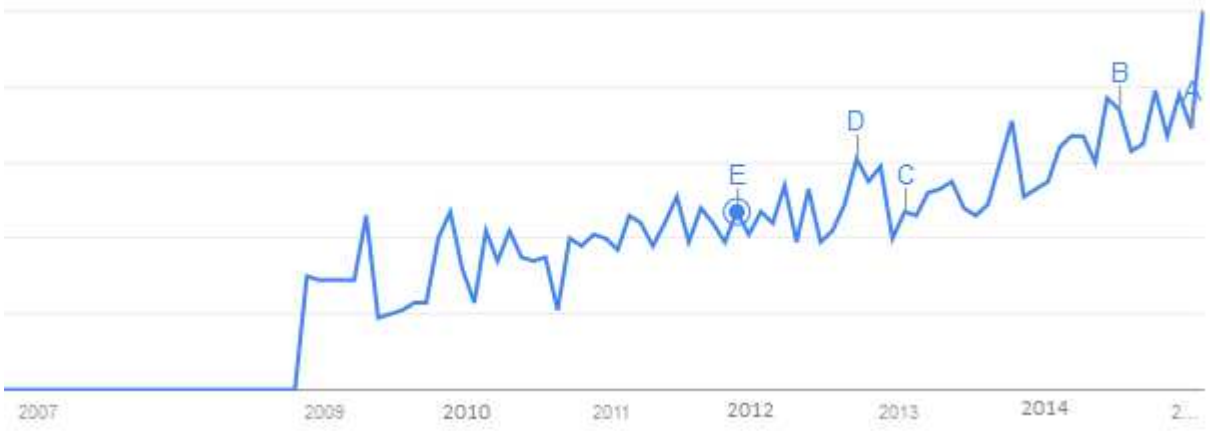

Figure 1. Search volume for "Technical Debt" on Google.com

Background

Knowledge representation systems are essential amid rising information production (Fogl 1979). This includes information that need to be shared during large information systems development. There are different tools that can be used to support the development of such type of systems including taxonomies

and ontologies. If on the one hand taxonomies allow organizing information and/or knowledge in hierarchical relationships between terms, on the other, ontology is a vocabulary representation that is often specialized to some domain or subject (Chandrasekaran et al. 1999). Thus, ontology supports the capture, representation, search, storage, and standardization of knowledge, describing a consistent, complete, and unambiguous vocabulary (Guarino 1998).

There are different classification schemes for ontologies. In one of them, their types are organized according to their level of generality in top-level, domain, task, and application (Guarino 1998). Other classification schema is based on their detail level. According to this schema, ontologies can be classified in lightweight and heavyweight (Calero 2006). In this work, the defined ontology is classified as lightweight domain ontology. Lightweight domain ontology considers the concepts of a particular domain, their properties, and relationships between them.

Despite its importance, to be useful, the ontology needs to be represented in a more intuitive way. One way to do this is to use knowledge visualization techniques. According to Tergan *et al.* (2006), knowledge visualization is a field of study that investigates the power of visual formats to represent knowledge with the purpose of supporting the cognitive process to generate, represent, structure, retrieve, and share knowledge. It is composed of a set of techniques and theories that help improving the way that knowledge is transferred between two or more people through visual features. Knowledge visualization uses techniques of information visualization as a tool to implement its concepts (Burkhard 2004).

The definition of ontology is a hard task mainly when it is expected it has relevance and value to a broad audience (Chen et al. 2008). In order to obtain better defined domain ontology, the collaboration between different experts needs to be stimulated allowing each of them to participate actively on its development (Rospocher et al. 2014; Jie et al. 2006). The use of collaborative systems can support the achievement of this objective. The central idea of collaborative systems is to support a group of people seeking similar goals to achieve these goals, so that these people help each other, directly or indirectly, through the system. According to Coleman (Coleman 1997), collaborative systems can be classified on the following types: content management, knowledge management, real time collaboration tools, virtual team tools, collaborative CRM, portals, and online communities. According to this taxonomy, the infrastructure proposed in this paper is classified as a Knowledge Management Collaborative System, because its main purpose is to collect (collaboratively), store, review, and provide knowledge on TD in an interactive and visual way.

Organizing the Knowledge on Technical Debt

Some initiatives have been performed in order to categorize the different types of debt. Fowler (2009) proposed the Technical Debt Quadrant where he classified the debts regarding their Reckless/Prudent and Deliberate/Inadvertent characteristics. McConnell (2007) organized the debt into two groups: intentional and unintentional. In another work, Tom *et al.* (2013) performed a multivocal literature review and a set of interviews with software practitioners and academics, and categorized the types of debt in a list of seven dimensions.

However, the aforementioned 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 where the debt was incurred or 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. Taking this into consideration, Alves *et al.* (2014) identified the following types of debt:

- Architecture Debt: Refers to the problems encountered in product architecture, for example, violation of modularity, which can affect architectural requirements (performance, robustness, among others). Normally this type of debt cannot be paid off with simple interventions in the code, implying more extensive development activities;
- Build Debt: Refers to issues that make the build task harder, and unnecessarily time consuming. The build process can involve code that does not contribute to value 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 build debt;

- Code Debt: Refers to the problems found in the source code that can negatively affect the legibility of the code making it more difficult to maintain. Usually, this debt can be identified by examining the source code for issues related to bad coding practices;
- Defect Debt: Refers to known defects, usually identified by testing activities or by the user and reported on bug tracking systems, that the Configuration Control Board (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 TD for a product, making it harder to fix them later;
- Design Debt: Refers to debt that can be discovered by analyzing the source code and identifying violations of the principles of good object-oriented design (e.g. very large or tightly coupled classes);
- 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;
- 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;
- Requirement 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: Refers to the inappropriate selection and substitution of web services that lead to worst code or architecture practices;
- Test Debt: Refers to issues found in testing activities that can affect the quality of those 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);
- Test Automation Debt: Refers to the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles. This debt can be considered a subtype of test debt.

As can be observed, TD can occur in different artifacts throughout the life cycle of a product. Even different instances of debt in the same artifact can be of different types. Moreover, the types of debt are not orthogonal, i.e. some instances of TD could conceivably fit into more than one category. For instance, the line that separates design and code debt is sometimes tenuous. Design debt is usually more concentrated on object oriented design practices. On the other side, code debt is more related to good coding practices. However, as the design is reflected in the code, we clearly have some overlapping between both types. More details on how the types of debt are related each other can be found in (Alves et al. 2014).

Associated with each type, the authors also specified its indicators. TD Indicators support the identification of TD items during information systems development. Once this information was collected, Alves et al. (2014) followed a process defined by Falbo *et al.* (1998) to develop the ontology. Thus, in the first moment, the competence (its use and purpose) of the ontology was defined. Its purpose was to organize the different TD types considering its nature as classification criterion. Next, the ontology capture and formalization was performed. This phase involved the identification and specification of concepts, their relationships, and all other elements necessary for the representation of the ontology. Finally, the proposed ontology was assessed in two steps. In the first, the quality criteria defined in (Gruber 1995) were considered. In the second step, an expert that did not participate of the ontology development performed an evaluation on it.

More recently, we have identified some causes that lead the development team on to incur technical debt. These causes were identified by an additional reading on the papers used to define the first version of the

ontology. The identified list of causes passed through an initial evaluation by five researchers. Only causes considered valid by three or more researchers were considered in the final list. At the end, we have evolved the ontology defined in (Alves et al. 2014) by adding causes for each type of debt. Table 1 represents the current version of ontology considering TD types, indicators, and causes.

TD Causes	TD Types	TD Indicators
- Inappropriate selection and definition of architecture and design patterns; - Lack of experience of the developers; - Low quality of code or application to quickly release the project.	Architecture Debt	- ACN/PWDR - Betweeness Centrality - Software Architecture Issues - Structural Analysis - Structural Dependencies - Violation of Modularity
- Lack of experience of the developers; - Low quality of code or application to quickly release the project.	Build Debt	- Structural Dependencies - Build Issues
- Insufficient investment in training; - Lack of experience of the developers; - Low quality of code or application to quickly release the project; - Violation of coding practices; - Lack of consistent definition for quality requirements.	Code Debt	- ASA Issues - Code Metrics - Code without Standards - Duplicated code - Multithread correctness - Slow Algorithm
- Defects accumulation over time; - Low quality of code or application to quickly release the project; - Differed defects and change requests to later releases.	Defect Debt	- Uncorrected known defects
- Project issues without refactoring; - Project complexity; - Lack of experience of the developers; - Low quality of code or application to quickly release the project; - Out of date/poor/insufficient documentation; - Inappropriate selection and definition of architecture and design patterns; - Accumulation of design pattern Grim and Rot.	Design Debt	- ASA Issues - Code Smells (Brain Method, Data Class, Data clumps, Dispersed Coupling, Duplicated Code, God class, Schizophrenic Class, Refused Parent Bequest, Intensive Coupling) - Code Metrics - Grime - Structural Analysis
- Requirements dependencies; - Tight deadlines; - Postponement of updating the documentation of the old features; - Increasing complexity of projects; - Misinterpretation; - Pressure to quickly solve customer problems; - Lack of understanding; - Lack of experience of documentation teams.	Documentation Debt	- Documentation Issues - Insufficient comments in code
- Lack of experience of the developers.	Infrastructure Debt	-
- Insufficient investment in training.	People Debt	-
- Lack of experience of the developers.	Process Debt	-
- Requirements dependencies; - Lack of experience of the developers.	Requirement Debt	- Requirement Backlog List
- Inappropriate selection/replacement of web services; - Lack of experience of the developers.	Service Debt	- Selection/Replacement of web service
- Tight deadlines; - Misinterpretation caused by complex or excessively dispersed instructions - Disperse instructions for planning / implementation of tests; - Using of wrong version of a document Insufficient investment in training; - Lack of experience of the developers; - Incomplete/inadequate test coverage.	Test Debt	- Incomplete Tests - Uncorrected Known Defects - Defects deferred
- Tight deadlines; - Lack of experience of the developers; - Incomplete/inadequate test coverage.	Test Automation Debt	-

Table 1. A view of the ontology of terms on TD considering its types, indicators and causes

The identification of possible causes for incurring TD in software projects can be considered an important step in order to have broader view on how the debt can be managed and, sometimes, prevented. However,

we still do not know how each cause is related to TD indicators. Some links between causes and indicators seems to be logically justified, for instance: lack of experience of developers and violation of modularity; lack of experience of developers and structural dependencies. However, the detailed investigation of this type of relationship (cause – consequence) can still be considered an open area and deserves empirical studies to support further analysis.

TD Wiki – Collaborative Sharing and Evolution of the Knowledge on Technical Debt

TD Wiki is a computational infrastructure that makes use of knowledge visualization techniques to support the collaborative sharing and evolution of knowledge on TD. Technical debt types, indicators, causes, and evaluation studies were organized to compose the knowledge base and can be evolved as follows:

- TD Type: inclusion of new types and identification of the most used by researchers and practitioners;
- TD Indicator: inclusion of new indicators and identification of those that have been considered more appropriate to find that kind of debt;
- TD Cause: inclusion of new causes and identification of those that more contribute to that kind of debt be incurred in the project;
- Evaluation Studies: inclusion of new studies that were conducted to assess a particular type of debt.

This allows: (1) researchers/practitioners to know the types of debt, as well as its indicators and causes, and those that have been most used or are more appropriate to be considered on project's activities; (2) researchers/practitioners to have an idea about the current knowledge level on a particular type of debt based on the empirical studies that evaluated it. This set of information can be useful when defining strategies for TD identification and monitoring during the development of information systems.

Requirements

The actors that will manipulate the infrastructure are:

- Unregistered user: refers to any visitor to the TD Wiki page on the Internet;
- Registered user: refers to researchers / practitioners who are registered on the infrastructure;
- Moderator: researchers / practitioners responsible for evaluating the information sent to TD Wiki before they are incorporated into the knowledge base, and;
- Administrator: responsible for defining the moderators.

This set of actors forms a hierarchy (administrator -> moderator -> registered user -> unregistered user) where higher hierarchical levels have access to the features of the lower levels. Table 2 presents the requirements defined for TD Wiki and the actors who have access to each of them.

Actor	Requirement
Unregistered User	**(REQ01)** To visualize the ontology of terms on technical debt;
	(REQ02) To visualize a summary (brief description and some of its indicators) of each type of debt;
	(REQ03) To visualize in detail the information about each type of debt including its definition, indicators, causes, evaluation studies, and references;
Registered User	**(REQ04)** To create a personal profile containing information about its work and research;
	(REQ05) To add new types of debt, indicators, causes, and references;
	(REQ06) To indicate, based on personal experiences, which indicators are more appropriate to support the identification of a particular type of debt;
	(REQ07) To indicate, based on personal experiences, which are the main causes that

	contribute to a type of debt be incurred;
Moderator	**(REQ08)** To evaluate the inserted information (new types, indicators, causes, and references);
Administrator	**(REQ09)** To define which users are moderators of the system.

Table 2. List of requirements

Architecture

The data model shown in Figure 2 was created to meet the functional requirements and to represent the organized knowledge in a relational schema. For this, entities were created to represent the main concepts (TD Types, Indicators, Causes, Evaluation Methods, and References) and the relationship between them.

Some of the collaborative aspects of the infrastructure were captured through tables such as *user_on_td* that stores the number of users that are researching and/or working on a particular type of debt. Besides, the columns relevance (in td_causes table), occurrence (in td_indicators table), and quantity (in td_evaluations table) allow tracking which causes and indicators are considered more relevant by registered users. Finally, the table td_translation was created to make possible to translate the mapped knowledge to other languages.

This data model was initially populated with the knowledge mapped in (Alves et al. 2014) and extended in this work (see "Organizing the Knowledge on Technical Debt" section).

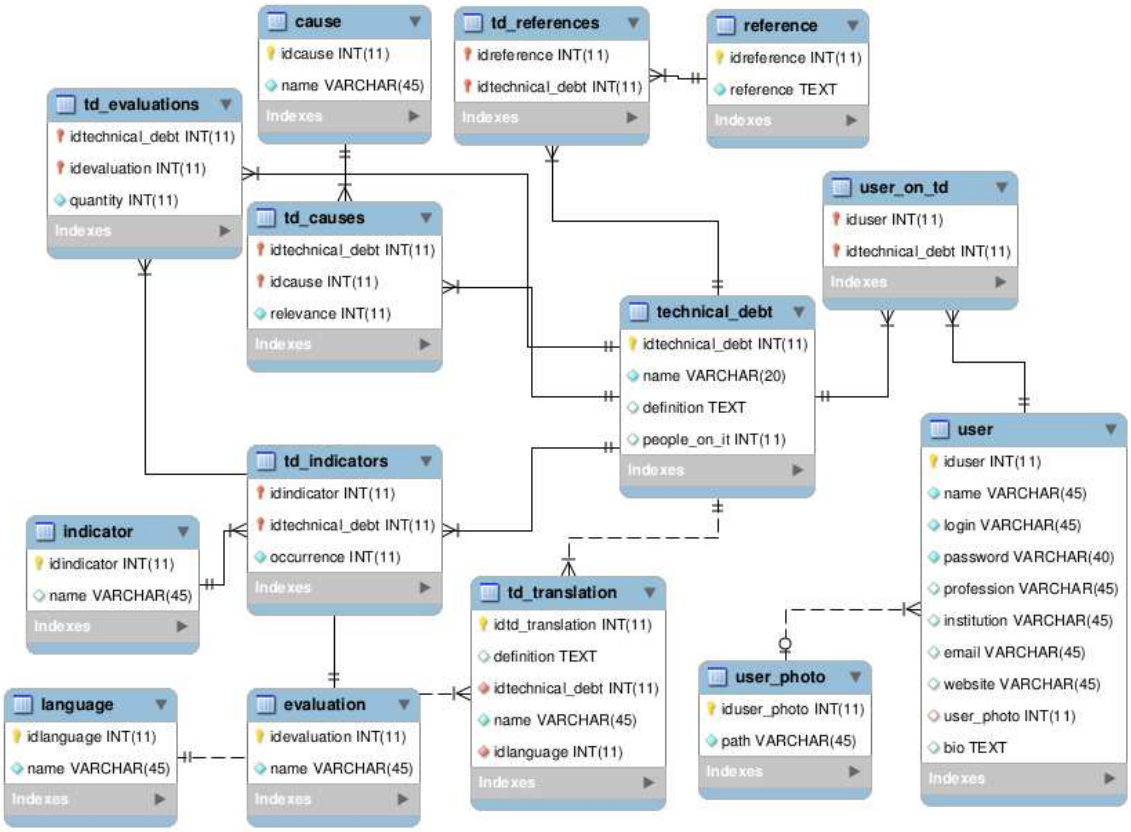

Figure 2. Data model of TD Wiki

Next, the four-layer architecture (FrontEnd, BackEnd, Data Persistence Layer, and Asynchronous Communication Layer) shown in Figure 3 was defined. In FrontEnd layer, we used the following technologies:

- JavaServer Pages (JSP): for rendering dynamic web pages;
- D3.js: library focused on providing rich experiences for information visualization (Bostock et al. 2011). This library was used to visually represent the knowledge on TD;
- JQuery: library used to provide a high abstraction of JavaScript and using of AJAX calls enabling the exchange of information with a low response time.

The using of these technologies together allowed us to follow the principles of User-Centered Design (Abras et al. 2004), which seeks to enrich the experience of use, and Data-Centered Design (Abras et al. 2004), which seeks to improve usability through a greater focus on how the data is presented. In addition, it was considered minimalist design concepts such as, for example, avoiding the use of unnecessary elements that could distract the user's attention of the relevant information.

The BackEnd layer was developed in Java JEE. The design of classes followed some design patterns such as Data Access Object (DAO), JavaBean, and Controller in order to facilitate the maintenance and future extensions of the code. TD Wiki also has an intermediate layer of asynchronous communication between the FrontEnd and BackEnd layers. This layer, based on the using of AJAX, enhances the user experience by reducing the response time because it does not require that a page is fully charged for each request. Moreover, because AJAX calls are effective when working with JSON objects and TD Wiki requires a smoother interaction when visually representing the knowledge on TD, part of the persisted information in MySQL database was replicated as JSON objects in JSON Files Repository.

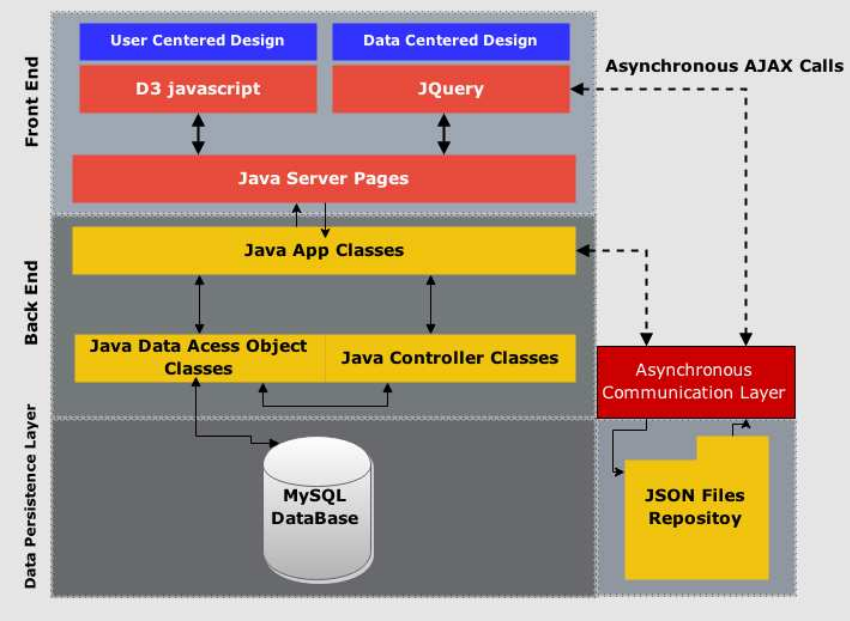

Figure 3. Architecture of TD Wiki

Finally, we have the data persistence layer that makes use of MySQL database system. All registered users' information is encrypted, protecting the data from unauthorized access.

The TD Wiki's Information Architecture (that defines how information can be accessed in the infrastructure) organizes the knowledge on TD in two levels: overview layer and detailed layer (Figure 4).

The overview layer is represented in the initial page of TD Wiki. On this page, an overview of the different types of debt will be showed containing, for each type, a brief description and some of its indicators. In addition, this page will contain an ontological tree representing the known types of TD. It is from this tree that the user can get more details about each type. In the detailed knowledge layer, the user has access to the known indicators of each type of debt, the causes that may lead to its occurrence, performed evaluation studies, and some references.

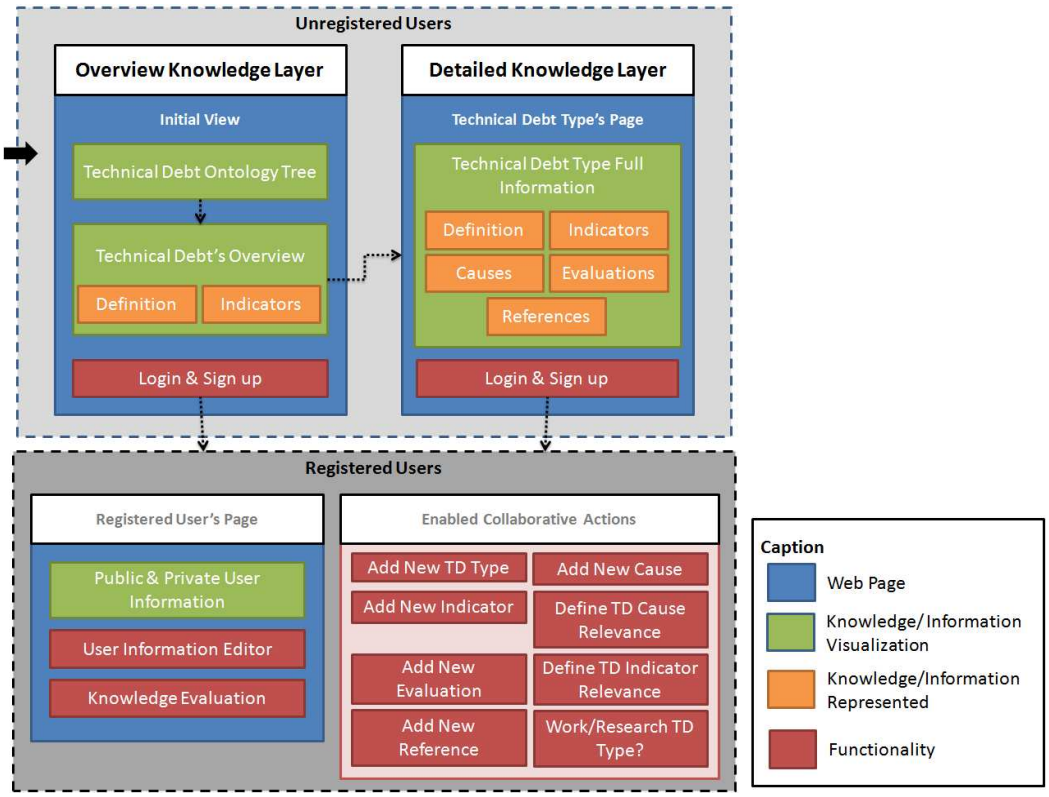

Figure 4. Information Architecture

When authenticated in the infrastructure, the users will have access to other features such as your personal page. They can contribute, based on their experience, to the evolution of the knowledge on TD by adding new types, indicators, causes, and also by the definition of the importance of each type, indicator, or cause.

TD Wiki

This section will present TD Wiki functionalities. They were grouped into three categories: Knowledge Sharing, Knowledge Evolution, and Supporting Features.

Knowledge Sharing

TD Wiki allows the sharing of knowledge on TD using knowledge visualization techniques. By accessing TD Wiki, the user has access to the initial view (Figure 5). In this page, it is possible to see the ontological tree that represents all types of TD (REQ01). Besides, the thickness of the lines connecting the types of debt to the root of the tree indicates how much that type has been considered in the monitoring of debt by the practitioners or investigated in research activities. To calculate this result, the users of the infrastructure will define the types of debt they are working or research on. Each vote will increment by 1 the total number of people working with that type of debt. Currently, the level of thickness represents the number of papers (identified in a mapping study that was performed by the authors of this paper and that is under submission process for a journal) that analyzed that type of debt.

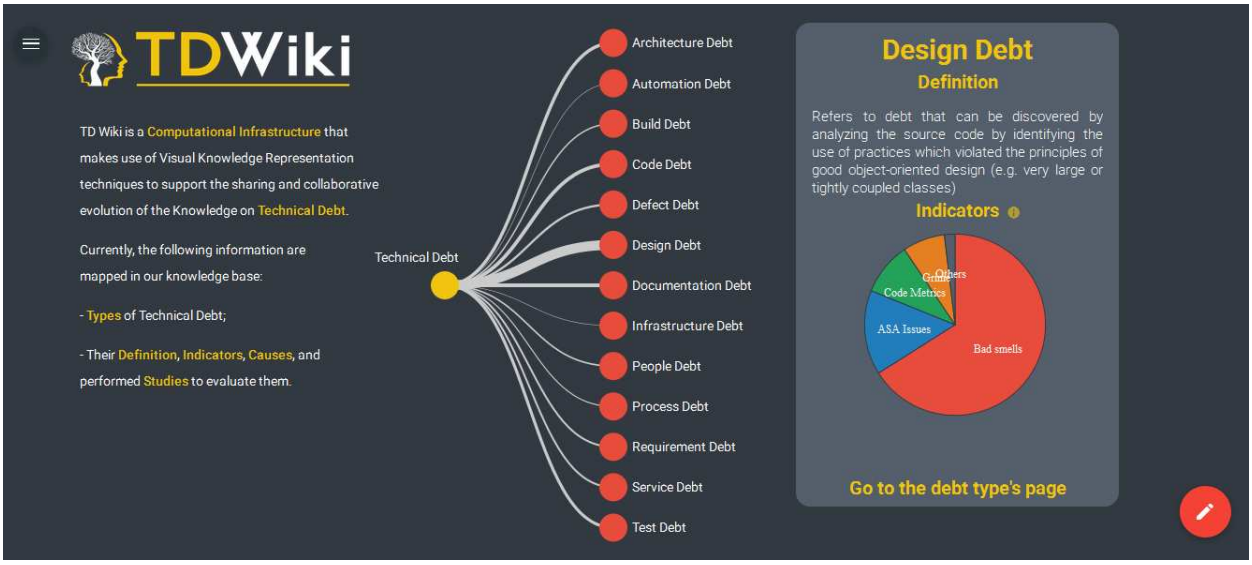

Figure 5. Initial View – Overview Knowledge Layer

When passing the mouse over a particular type, a brief summary of that type will be presented containing its definition and some of its indicators (REQ02). At the end of this summary, there is the 'Go to the Debt Type's Page' option that directs the user to the corresponding Technical Debt Type's Page (Figure 6) (REQ03). On this page the user can access detailed information about that kind of debt:

- TD type definition: contains the definition of the type;
- TD indicators: contains indicators that have been proposed to support the identification of debt items in software projects. The user can also find out which indicators are more recommended to support the task of identifying that kind of debt (the higher it appears in the list, more recommended it is). In this area, TD Wiki also has a pie chart that shows the proportion of recommendations made by users. Currently, the graph represents the number of papers (identified in a mapping study that was performed by the authors of this paper) that cited each indicator;
- TD Causes: contains possible causes that lead the development team to incur that type of debt. The Ishikawa Diagram was used to represent the identified causes and their consequence. In this diagram, as thicker is the line associated with the cause, more relevant the cause is considered as a factor that contributes to the insertion of that type of debt in software projects. Currently, the level of thickness represents the number of papers (identified in a mapping study that was performed by the authors of this paper) that cited each cause;
- Evaluation Studies: represents studies that evaluated that type of debt. The number of studies on a particular type can be an indicator of how much we know about it;
- References: contains a list of references from which this information was extracted.

Knowledge Evolution

TD Wiki allows the collaborative evolution of knowledge on TD, enabling registered users to add new types of debt, new indicators, new causes, and new references (REQ05). For each of those add-functionalities, a group of three moderators assess the provided information. If approved, the new information is incorporated into the TD Wiki's knowledge base. This control is necessary because TD has been sometimes used as a buzzword, which can lead to erroneous interpretations of its meaning.

In addition, TD Wiki also allows users to indicate what type they are working on and, based on their experience, which indicators are more recommended for the identification of a particular type of debt (REQ06), and what are the main causes that contribute to a type of debt be incurred (REQ07). In Figure 7, frame A, the user can signal that is working/researching on a particular type of debt. The result of this action will be presented on the ontological tree. In frame B, the user can indicate new studies that evaluated that type by specifying the type of study and its references.

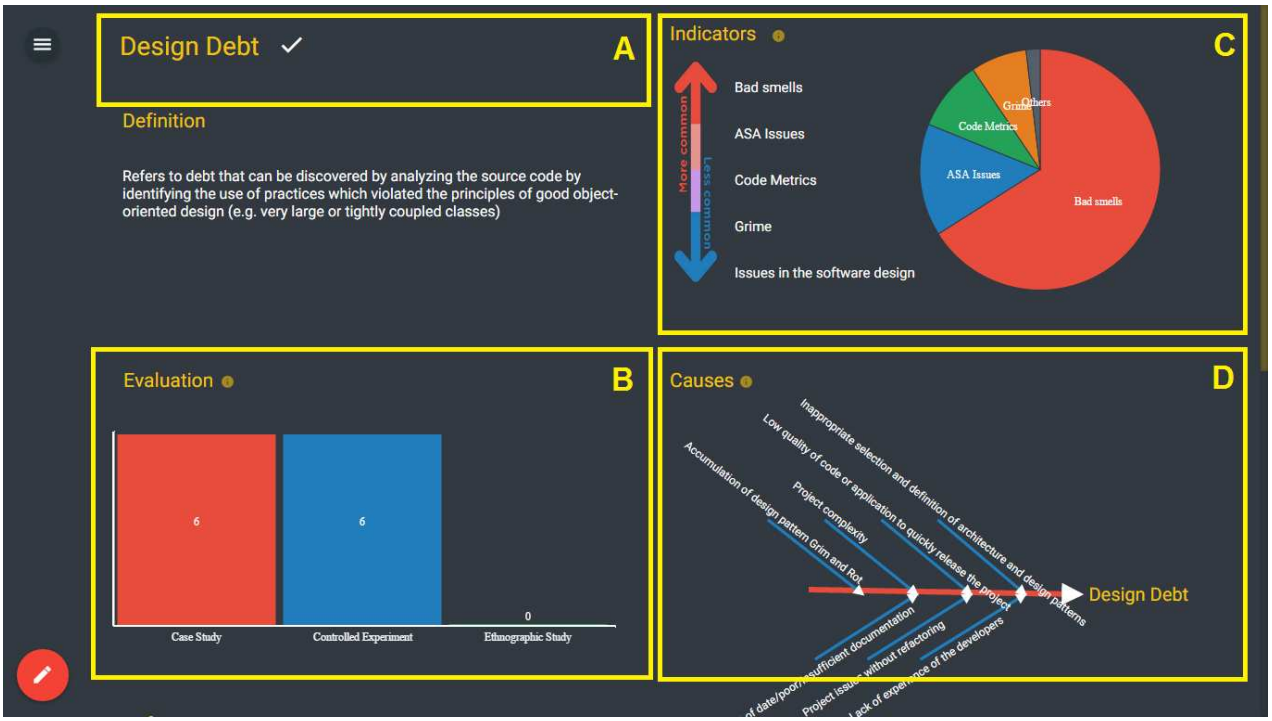

Figure 6. Technical Debt Type's Page – Detailed Knowledge Layer

In frame C, the user can indicate the relevance of a particular TD indicator. For this, the user moves up/down the indicators in accordance with the scale defined by the arrow on the left side of the figure. As result, the pie chart will be updated considering the ranking defined by all users of the infrastructure.

Finally, in frame D, the user can indicate the causes that more lead to the occurrence of a particular type of debt. In doing so, TD Wiki will recalculate the weight that each cause has and will represent this information by showing the connecting lines in different thickness.

Supporting Features

Access control will allow the registration of new users and their access to collaborative functionalities of TD Wiki. Registered users can configure their personal profiles, containing information about their work and research, full name, affiliation, profession, email, and website (REQ4). The user can also visualize its recent activities (performed collaborations), change the password, and send e-mail to TD Wiki team. The idea of having a social profile within the infrastructure strengthens the concept of collaborative systems as it will map the practitioners and researchers who have been working with technical debt.

Besides, a group of three moderators defined by the administrator of TD Wiki (REQ09) will be able to evaluate new information provided by users (REQ08) (Figure 7). To be included in the knowledge base of TD Wiki, the information needs to be approved by at least two moderators. In the case of the information do not be approved, the moderator needs to justify its decision. At the end, the user who requested the inclusion will be informed about the result of its request.

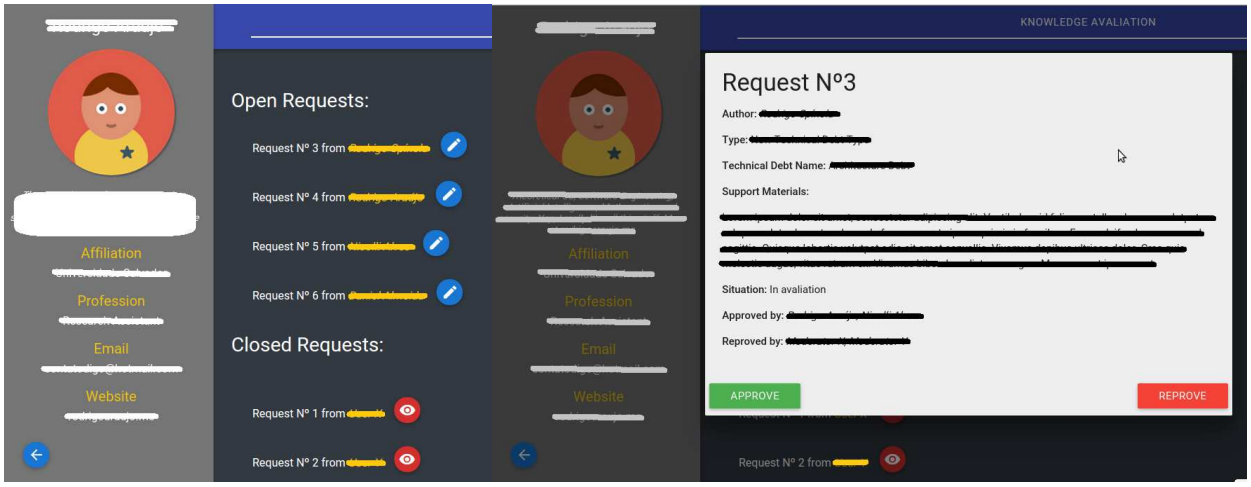

Figure 7. Knowledge evaluation page

Final Remarks

The development of information systems has a series of challenges. The accumulation of technical debt creates an additional difficulty impacting system evolution activities. Identifying and monitoring the debt is essential to reduce the risks involved in the construction of these systems. However, even before the development team can work on the identification and monitoring of TD, it is necessary to understand what types of debt can be inserted, how it can be identified, and which causes can lead the development team to incur them into the project.

In this context, this work presented TD Wiki, a computational infrastructure for collaborative sharing and evolution of knowledge on TD. As main contribution, TD Wiki provides necessary information for development teams start to monitor TD on their projects. This can be considered an important step in order to improve the quality of information systems under development. Besides, TD Wiki creates a channel that stimulates the discussion on TD by practitioners and researchers, and allows the collaborative evolution of this knowledge.

The knowledge represented in the infrastructure is based on the work of Alves et al. (2014) and its extension in this research. There is no guarantee that all types, indicators and/or causes are currently represented. However, the infrastructure is already available for use, and thus, it is expected that the mapped knowledge is evolved in the course of time.

Currently, we are working on a qualitative study to investigate in more details the TD causes. As others next steps of this research, TD Wiki will be evaluated with respect to their usability and ability to represent the knowledge on TD. We also intend to continuously improve TD Wiki's functionalities considering suggestions sent by users.

Acknowledgements

The authors would like to thank CAPES and FAPESB for the financial support to this work. This work was also partially supported by CNPq Universal 2014 grant 458261/2014-9.

REFERENCES

Abras, C., Maloney-Krichmar, D., Preece, J. 2004. "User-Centered Design". In Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications.

Alves, N. S. R., Ribeiro, L. F., Caires, V., Mendes, T. S., and Spínola, R. O. 2014. "Towards an Ontology of Terms on Technical Debt". In Proceedings of the 2014 Sixth International Workshop on Managing Technical Debt (MTD '14). IEEE Computer Society, Washington, DC, USA, 1-7. DOI=10.1109/MTD.2014.9 http://dx.doi.org/10.1109/MTD.2014.9

Bostock, M., Ogievetsky, V., and Heer, J. 2011. D3 Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics 17, 12 (December 2011), 2301-2309. DOI=10.1109/TVCG.2011.185 http://dx.doi.org/10.1109/TVCG.2011.185

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., and Zazworka, N. 2010. "Managing technical debt in software-reliant systems". In Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER '10). ACM, New York, NY, USA, 47-52. DOI=10.1145/1882362.1882373 http://doi.acm.org/10.1145/1882362.1882373.

Burkhard, R. A. 2004. "Learning from architects: the difference between knowledge visualization and information visualization" in IV '04: Proceedings of the Information Visualisation, Eighth International Conference, pages 519–524, Washington, DC, USA. IEEE Computer Society.

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

Chandrasekaran, B., Josephson, J. R., Benjamins, V. R. 1999. "What Are Ontologies, and Why Do We Need Them?". IEEE Intelligent Systems 14, 1 (January 1999), 20-26. DOI=10.1109/5254.747902 http://dx.doi.org/10.1109/5254.747902.

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

Coleman, D. 1997. "Groupware: Collaborative Strategies for Corporate Lans and Intranets" in Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

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

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

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

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

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

Guo, Y., Spínola, R.O., Seaman, C., 2014. "Exploring the costs of technical debt management - a case study" in Empirical Software Engineering Journal, v.1, p.1 - 24. DOI:10.1007/s10664-014-9351-7

Izurieta, C., Vetró, A., Zazworka, N., Cai, Y., Seaman, C., Shull, F. 2012."Organizing the Technical Debt Landscape". In Proceedings of the Third International Workshop on Managing Technical Debt (MTD '12). IEEE Press, Piscataway, NJ, USA, 23-26.

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

Kruchten, P., Nord, R. L., Ozkaya, I. 2012. "Technical Debt: From Metaphor to Theory and Practice". IEEE Software. 29, 6 (November 2012), 18-21. DOI=10.1109/MS.2012.167 http://dx.doi.org/10.1109/MS.2012.167.

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

Pfleeger, S. L. and Atlee, J. M. 2009. "Software Engineering: Theory and Practice" in 4th Ed. Prentice Hall.

Rospocher, M., Ghidini, C., Di Francescomarino C. 2014. "Evaluating Wiki Collaborative Features in Ontology Authoring" in IEEE Transactions on Knowledge and Data Engineering, IEEE computer Society Digital Library. IEEE Computer Society.

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

Spínola, R. O., Zazworka, N., Vetró, A., Seaman, C. and Shull, F. 2013. "Investigating technical debt folklore: Shedding some light on technical debt opinion". In Proceedings of the 4th International Workshop on Managing Technical Debt (MTD '13). IEEE Press, Piscataway, NJ, USA, 1-7.

Tergan, S.O., Keller, T., and Burkard, R. A. 2006. "Integrating knowledge and information: digital concept maps as a bridging technology" in Information Visualization, 5(3):167–174.

Tom, E., Aurum, A., and Vidgen, R. 2013. “An exploration of technical debt” in Journal of Systems and Software. 86, 6, 1498-1516. DOI=10.1016/j.jss.2012.12.052

Zazworka, N., Spínola, R. O., Vetró, A., Shull, F. and Seaman, C. 2013. “A case study on effectively identifying technical debt”. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13). ACM, New York, NY, USA, 42-47. DOI=10.1145/2460999.2461005 http://doi.acm.org/10.1145/2460999.2461005.