TD-Manager: a tool for managing technical debt through integrated catalog

Lucas Borante Foganholi, Rogério Eduardo Garcia, Danilo Medeiros Eler, Ronaldo Celso Messias Correia, Celso Olivete Junior
Faculdade de Ciências e Tecnologia, UNESP – Univ Estadual Paulista, Presidente Prudente, Departamento de Matemática e Computação
Presidente Prudente/SP, Brazil
E-mail: borante@gmail.com, {rogerio, daniloeler, ronaldo, olivete}@fct.unesp.br

Abstract—Technical debt is an emergent area that has stimulated academical concern, its practical application cope development activities such as documentation, design, code and test. However, literature review pointed out an integration gap between identifying and accurately cataloging technical debt. It also mentioned bunch of tools for most activities on software development process that could identify technical debt, but there is not a described solution that supports cataloging all types of debt. In this context, this work aims to present an approach to catalog technical debt related to any activity mentioned, tabulating and managing its properties. This catalog is implemented by TD-Manager tool, that allows register technical debt and control debt status. In addition, the tool can integrate with technical debt identification tools and import debt to catalog. In order to show the approach, we present an integration of technical debt identified using Sonar, mapping relationship and managing external information integrated.

Keywords: Technical Debt, Debt Cataloging, Integration on Debt Identification, Technical Debt Management, Technical Debt Cataloging Tool

I. INTRODUCTION

The term “technical debt” is used as a metaphor to refer to the likely long-term costs associated with software development and maintenance shortcuts taken by programmers to deliver short-term business benefits [1], [2], [3]. During software development is usual to prioritize some activities, leaving others in background, such as defect correction and documentation. The main reason for that is to reach the expectations imposed due to time or financial constraints. Thereby, postpone a technical debt creates technical debt.

Technical Debt (TD) provides useful guidance when a trade off of code quality must be made against another factor, e.g., delivering functionality more quickly to achieve time to market objectives [4]. Seaman and Guo [5] explain that software managers, to use that trade off, need to balance the benefit of incurring TD with the associated costs when planning their projects, taking decisions supported by information. Although, the lack of information is a problem.

Letouzey and Ilkiewicz [6] described some of these manager actions such as setting targets for debt and specifying what level and which debt types are acceptable for the project or organization; analyzing and understanding debt to estimate potential impact and provide rationale for decisions; using TD as input for governance of application assets and analyzing an application’s debt in correlation with other information such as business value or user perceived quality; and institutionalizing the previous practices and putting in place tools and processes to produce the benefits of proactive TD management. To assist in those activities, there are techniques (such as code smells, design patterns, test results for defect debt and comparing with test plans for testing debt) and tools (such as FindBugs, Sonar, code coverage tools) that could potentially be useful in the identification of TD, even if many of them were not developed for that [7].

In this scenario an existent problem consists in catalog TD items independent of the debt type, and use contextualized information when the debt is detected. Thus, the goal of this paper is to present a tool named TD-Manager, which aims to create a catalog integrating TD from design/code, test, documentation, defect and infrastructure activities. It also intends to integrate with database of identification tools in order to import technical debt.

The methodology used in this paper consists in extending the described contents of TD items proposed by Seaman and Guo [5], creating an integrated catalog, for managing TD of any type, in a semi automatic process. TD-Manager is a proposed tool that keeps records of technical debts, as described in this paper. The integration made between TD-Manager and Sonar allows catalog and manage the information after TD identification accomplished in the second tool.

This paper is organized as follows: in Section II is presented a background used in the work, in Section III is detailed the proposed approach, in Section IV is described TD-Manager tool which implements the proposed approach, in Section V is demonstrated a case of study using the tool integrated with Sonar, in Section VI is presented a discussion about the proposed approach and in Section VII is presented the conclusion and future works.
II. BACKGROUND

A. Identification of Technical Debt

Guo and Seaman [8] proposed a classification of technical debt into four main types: design (or code), testing, defect and documentation debt. In each type of TD is possible to find techniques and tools to identify the debt.

Design or code debt can be identified by statically analyzing source code or inspecting code compliance to standards [5]. Izurieta et al. [7] mentioned techniques and tools that could potentially be useful in the identification activity. They presented four specific identification techniques: code smells, automatic static analysis, modularity violations and design patterns, described in the following.

Code smells (a.k.a. bad smells), which the concept was introduced by Fowler [9], describes choices in object-oriented systems that do not use principles of good-object oriented design. Automatic approaches have been developed to identify code smells, as proposed by Marinescu [10]. Former studies have shown that some code smells are correlated with defect- and change-proneness [11]. In context of code smells technique, an example of tool is CodeVizard [11].

Automatic static analysis (ASA) is a reverse engineering technique that consists in extracting information about a program from its source code using automatic tools [12]. ASA tools search issues based on violations on recommended programming practices and potential defects that might cause faults or degrade some dimensions of software quality such as maintainability, efficiency, among others. Some ASA issues can indicate TD as they are good candidates for removal through refactoring to avoid future problems. In previous work Vetro et al. [13] analyzed the issues detected by FindBugs on two pools of similar small programs. Some of the issues identified as good/bad defect detectors by [13] were also found in similar studies with FindBugs, both in industry [14] and open source software [15]. Some studies have also been conducted with other tools (e.g. Sonar) and the overall finding is: a small set of ASA issues is related to defects in the software, but the set depends on the context and type of the software.

Izurieta et al. [7] described a context for modularity violations and how it connects with technical debt. During software evolution, if two components always are changed together to accommodate modifications but they belong to two separate modules designed to evolve independently, then there is an unconformity. Such unconformity can indicate TD, as they may be caused by side effects of a quick and dirty implementation, or requirements might have changed such that the original designed architecture could not be easily adapted. When such discrepancies exist, the software can deviate from its designed modular structure, which is called a modularity violation. Wong et al. [16] have demonstrated the feasibility and utility of this approach. In their experiment using Hadoop, they identified 231 modularity violations from 490 modification requests and 152 (65%) violations were conservatively confirmed. For modularity violations, another example of tool is CLIO [16].

Design patterns [17] are popular because, and the reasons are not limited to, it claims to facilitate maintainability and flexibility of designs, to reduce number of defects and faults, and to improve architectural designs. Software designs decay as systems and operational environments evolve, and it can involve design patterns. Classes that participate in design pattern realizations accumulate grime non-pattern related code. Grime represents a form of TD, since the effort to keep the patterns cleanly instantiated has been deferred. In prior studies Izurieta and Bieman [18] introduced the notion of design pattern grime and performed a study on three opensource systems, JReactory, ArgoUML and eXist.

The four described techniques focus on source code or design TD type. There are other types of TD such as testing, defect and documentation, that will be contextualized in the following.

Testing debt are tests that were planned but not implemented/executed or got lost, it is also test cases not updated for new/changed functionality or low code coverage [5]. They can be identified by comparing test plans to their results, created on planning tests and not executed and, again, when identified low code coverage [5].

Most testing tools mentioned are not planned to identify TD, but it can be used in explained situation. Yang and Li [19] survey Coverage-based tools and compared 17 tools based on three features: code coverage measurement, coverage criteria and automation and reporting. Shah and Torchiano [20] present through systematic review the consequences of exploratory testing as TD. Exploratory testing is an approach that does not rely on the formal test case definition – instead of designing test cases, the execution and evaluation of the software behavior are based on tester intuition and knowledge.

Documentation debt are documentation that is not kept up-to-date [5], such as API documentation, requirements and use case documentation. They can be identified by comparing change reports to documentation version histories. If modifications are made without accompanying changes to documentation, the corresponding not updated documentation is documentation debt [5]. Forward and Lethbridge [21] identified, through a survey, tools used to deal with documentation in software projects, including automated generation of documentation.

Defect debt are known defects that are not yet fixed [5], such as low priority or severity defects due to rarely manifest or presented workarounds. They can be identified by comparing test results to change reports, the defects found and not fixed are defect debt items [5]. The tools used to find source code debts and the tools that support test executions are capable to identify defects. Snipes and Robinson [22] detailed a technique based on change control
boards (CCB) to categorize and prioritize defects supporting manager decisions to fix/defer debts based on cost-benefit analysis.

B. Cataloging Technical Debt

According to Seaman and Guo [5], the management of TD is based on a TD list, which is similar to a task backlog. The backlog contains TD items (in the following simply referred to as items), each one represents a task left undone that incurs a risk of causing future problems if not completed. Each item includes a description of what part of the system the debt item is related to, why that task needs to be done, and estimates the TD’s principal and interest, as well as other attributes described in their work.

Initially, when an item is created, the principal, expected interest amount, interest standard deviation and correlations with other debt items can be estimated subjectively according to the maintainer’s experience [5]. Since it is uncertain whether extra effort will be required, they used expected interest amount and standard deviation to capture the uncertainty. These rough estimation can be adjusted later using historical data or other types of program analysis.

III. THE PROPOSED APPROACH

In this section is described the proposed approach which has three activity groups: Identification, Cataloging and Managing, as depicted in Figure 1

![Figure 1: Proposed Approach](image)

The identification group initiates the proposed approach and copes technical debt identification activities for the types design (or code), testing, documentation and defect. The result or output of this activity is a list containing candidates to TD. As exposed in the previous section, there are techniques and tools for identification of each kind of technical debt mentioned.

Technical debt cataloging is the second group in the proposed approach and initiates after identification. This group has two activities: manual cataloging and semi automated cataloging, explained in the following. Both activities must be performed by a user, in the role of collector. The output of cataloging is a set of register of technical debt.

Manual cataloging is the activity to catalog any technical debt manually, independent of debt origin or way of identification it can be registered in this activity.

Semi automated cataloging uses the candidates of TD list generated in the previous group as input and, after analysis, catalog the item from the TD list. The collector analyzes the list and, for each TD item contained, reject or register as TD item, fulfilling the required information such as Date, Responsible, Type, Description and the estimation attributes, if not defined.

For both described activities of cataloging group is necessary specify the integrated catalog which explains the documentation structure for technical debt. The structure for technical debt catalog is based onto the item structure described in subsection II-B proposed by Seaman and Guo [5]. This structure is extended in the proposed approach in order to properly present technical debt at the management level. Item structure are described in Table I.

![Table I: Technical Debt Template - Adapted from [5]](image)

<table>
<thead>
<tr>
<th>ID</th>
<th>TD identification number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Date of TD identification</td>
</tr>
<tr>
<td>Responsible</td>
<td>Person or role which should fix this TD item</td>
</tr>
<tr>
<td>Type</td>
<td>design, documentation, defect, testing, or other type of debt</td>
</tr>
<tr>
<td>Project</td>
<td>Name of project or software application</td>
</tr>
<tr>
<td>Location</td>
<td>List of files/classes/methods or documents/pages involved</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the anomaly and possible impacts on future maintenance</td>
</tr>
<tr>
<td>Estimated Principal</td>
<td>How much work is required to pay off this TD item on a three point scale: High/Medium/Low</td>
</tr>
<tr>
<td>Estimated Interest Amount</td>
<td>How much extra work will need to be performed in the future if this TD item is not paid off now on a three point scale: High/Medium/Low</td>
</tr>
<tr>
<td>Estimated Interest Probability</td>
<td>How likely is it that this item, if not paid off, will cause extra work to be necessary in the future on a three point scale: High/Medium/Low</td>
</tr>
<tr>
<td>Intentional</td>
<td>Yes/No/Don’t Know</td>
</tr>
<tr>
<td>Fixed By</td>
<td>Person or role who really fix this TD item</td>
</tr>
<tr>
<td>Fixed Date</td>
<td>Date of TD conclusion</td>
</tr>
<tr>
<td>Realized Principal</td>
<td>How much work was required to pay off this TD item on a three point scale: High/Medium/Low</td>
</tr>
<tr>
<td>Realized Interest Amount</td>
<td>How much extra work was needed to be performed if this TD item was not paid off at moment of detection, on a three point scale: High/Medium/Low</td>
</tr>
</tbody>
</table>
or changing TD status. TD monitoring activity is used to track TDs life-cycle, due dates and, when some changes are needed, it can use previous activity to perform any modification.

IV. TD-MANAGER TOOL

TD-Manager is a tool that implements the approach (protocol) described in this paper. TD-Manager uses the integrated catalog as metadata in order to register technical debt properties. It is was developed using Java language and public community maintained frameworks to improve development productivity and reuse. The tool was developed using MVC (Model View Controller) design pattern and the frameworks used for persistence (or model) and interface (or view) layers are Hibernate and Vaadin, respectively.

It is a web application with an internal Tomcat as web container, allowing to execute the web application archive (war) file as a Java archive (jar), and as a web-based software the advantages as detailed in [23]. Since TD-Manager uses Vaadin 7, the web browsers supported by this version of the framework is also supported by the tool, but it was intensively tested using Google Chrome browser.

In the model layer, TD-Manager uses the proposed integrated catalog (or TD catalog) as an entity and its attributes as columns. It also has an entity for users (a.k.a responsible), one entity to register the connection information and one for mapping the relationship between TD-Manager and external applications. In the current version the tool only supports Postgres as its database and MySQL, Oracle, Postgres and SQL Server as external database.

The generic layout of the application contains a left side steady menu used to navigate among the programs (or functionalities). Each program represents one or more activity described on previous section. The menu also contains the logged user information and application logout.

The architecture of the solution is depicted in Figure 2 and contains Integration, TD Cataloging, Authentication and User components.

Integration correspond to external integration process, allowing to configure, test and save the information needed to connect. The first step to use this functionality is to set the connection integration properties which comprehends Alias, Server, Port, Database Name, Database Type, User and Password. After inform all the required fields it is possible to test the connection. This is the most difficult configuration because demands knowledge on identification tool database information and credentials, and TD-Manager, in its actual version, only allows direct database connection.

After configuring the connection it is indispensable to map a relationship between TD catalog fields and identification tool properties. For this mapping the user needs to provide external table name, external field and TD-Manager catalog field, establishing the relationship. In the end of this process the user can check results of the integration map and retrieve foreign TD candidates.

Authentication and User component corresponds to application authentication and permissions, associated to profiles, to access the functionalities. The user is also used when associating a TD to a responsible or fixed by person, which creates a relationship between User component and TD Cataloging component.

TD Cataloging corresponds to the register of technical debt and can be inputed manually or retrieved after integration synchronization through a foreign database connection from a TD identification tool, detailed in the following. This component also contains manager functionalities such as TD monitoring, control and assignment. It is possible through filters manipulation, attributes sorting and status controlling.

For manual TD cataloging is only mandatory to fulfill the TD information as described in the extended catalog proposed in previous section, and no integration or extra application is used. The collector, with an external TD candidates list, can add TD after being logged into application. For this catalog it is mandatory complete the required attributes and the tool validates it, not allowing to save if they are not detailed.

For semi automatic interface it is necessary to create a database integration and mapping the relationship between TD-Manager and the external tool. The interaction between TD Cataloging and Integration component is viable only after configure connection information and create the relationship between the fields of the integrated catalog and external database table, as explained in Integration component. With this, it is possible to use TD-Manager interface to filter and sort the external TD identified by the integrated tool and import to TD-Manager. This external filter and sorting is a mechanism used by the controller to help on TD candidates list analysis and only the selected ones are cataloged. At this moment, is possible to override some mapped (or not, which
means null information from external identification tool) information such as responsible, TD type and the estimated fields.

In the semi automatic interface was also created an attribute to register external information ID, avoiding duplicate the cataloged TD and alerting when the external key is marked for integration twice. For the appropriate use of this functionality it is vital to proper configure the external field ID in the mapping properties. After mapping the relationship between these both tools for each property of the catalog, it is possible to check the results of configured properties (or mapping). This functionality allows the user, responsible to define the relationship, to verify external tool data in TD-Manager and change it if the information is not accurate.

In both options, it is only possible to associate any TD register with a responsible after include this person as a application user. While TD is not finished or deleted, it remains available to manager deal with TD relevance, change estimated effort to fix, debt interest or probability to cause extra work in the future. It also can control the date when TD was fixed and the realized efforts to fix it.

The implementation of integrated catalog and an overview of the tool is presented in case study.

V. Case Study Using TD-Manager and Sonar

In this case study is presented the use of TD-Manager to catalog TD from open source project named OpenRefine [24], which is a tool for working with messy data: cleaning it; transforming it from one format into another; extending it with web services; and linking it to databases. The project is being developed using Java and contains 36,883 LOCs (Lines of Code) distributed in 496 classes, a wiki as documentation [24] and some issues cataloged on Github repository.

Due to space constraints only one TD identification tool to perform the integration is presented. Sonar is a world wide adopted (ASA) tool to analyze source code as mentioned in background section. The tool is based on rules to identify source code defects or refactoring points and when integrated with FindBugs, in the version 3.7.4 and for Java projects, it contains a total of 516 rules. These rules are classified in severities and include bad practices, correctness and performance issues, design flaws, code issues, security, etc.

The issues are identified by Sonar (TD candidates) and to show the use of TD-Manager for integrate a TD catalog, it is necessary to create a connection between TD-Manager and Sonar. Since Sonar was locally installed and configured to use Postgres database as it issues repository, the connection in TD-Manager is created using Postgres database, localhost server and Sonar credentials, as presented in Figure 3. In the Figure 3 it also possible to see how TD-Manager creates the relationship, explained in the following.

After successfully test the connection, it is necessary to map the external table and fields to import TD candidates list. In this step is necessary to know the characteristic of the Sonar, which table resides the main information and the specific fields related to the proposed catalog. From Postgres database investigation on Sonar schema it is possible to find “Issues” table, which contains all identified information for OpenRefine project. Using the table “Issues” it is possible to create the relationship between the two application with fields as shown in Table II.

After mapping the relationship between these both tools for each property of the catalog, TD-Manager extracts the information through Integration Sync interface and the manager can filter or sort this information to register technical debt into the integrated catalog. With the information inserted into the catalog, managers is allowed to take control of it through the Technical Debt interface, which is the same to manual register TD.

In Figure 4 is presented an example of an integrated technical debt of design type assigned to the responsible named Administrator. The debt occurred on project OpenRefine in line 83 of class com.google.refine.browsing.facets.ListFacet
and was registered with low estimated principal, which means low difficult to fix, and medium interest amount, which means medium difficult if decided to postpone implementation to fix it. It also possible to retrieve information inserted after fix TD such as date when the implementation finishes, author who fixed the TD and the real effort to fix (realized principal), remaining low – same as in estimation.

VI. DISCUSSIONS

It is possible to find several works related to decision making process in TD, such as [25], [26], [22]. TD-Manager tool aims to help managers to control TD and support their decision, establishing a closer relationship to those works. But besides the fact that integrated catalog is an extension of previous work [5] and includes additional information, it is possible that in the future more information for the integrated catalog is needed.

TD-Manager may cause accumulation of technical debt. The integrated catalog are designed to be complete and intuitive but it cannot be fast enough in order to capture all information of technical debts and update them. Just the needed to use another tool to manage the debt could make the responsible of the debt to not update the information.

It is being developed another work to refine and expand the approach proposed in this paper. The goal of this future work is aimed to improve an automated process of identifying technical debt and use a detailed catalog with contextualized information of identified technical debt items.

VII. CONCLUSIONS

The tool is designed to facilitate the integration between different types of TD. It is possible since unique repositories of all kind of debts are the key to group all information related to TD. Considering this design, TD-Manager focused on easily creates database integration, supporting the four most popular database engines [27], and simply assists the relationship mapping between the tools.

The approach produces and maintains a TD list according to the captured TD. TD List provides valuable information to existing software components. It can be helpful for generate agile backlog list, can provide a list of needed updates on documentation, can help on prioritize defects and can evidence faults on testing area.

It was presented a case study with one kind of technical debt (source code/design) and the tool was not yet integrated with testing, defect and documentation TD identification tools. Although, the integration is based on database direct connection, which means if any TD identification tool provides the connection credentials it is possible to integrate using the same steps described in the case study.

TD-Manager also can capture human-detected technical debt. In addition to the integration, it is possible to catalog any kind of technical debt identified by stakeholders. Since they are fully aware of all active requirements and development conventions, they can find additional technical debts, not identified by any tools. This ensures that information regarding the any kind of integration is possible to be cataloged.

By grouping technical debt identification from different project development level (code, test, and document), TD-Manager tool ensures that development is conducted while aware of technical debt’s presence. That allows any stakeholder to avoid unintentionally increasing the value of technical debt through dependencies to affected areas and to decrease efficiently its value by tackling technical debt in areas where the project is currently conducted.

TD-Manager allows multiple TD indicators to be used instead of only one of the mentioned tools and it is strongly recommended, since different tools point to different problems in a software artifact. The use of a single tool or single indicator (e.g. a single code smells) will only in rare cases point to all important TD issues in a project. As a result, project teams need to make intentional decisions about which of the TD indicators are of most relevance to them, based on the quality goals of their project, as suggested in [?].

After apply TD-Manager on a small software development company for Java developers and tester teams, the project manager reported positive results in adoption of the tool. He stated that, besides the only three option to control estimated and realized efforts, it was possible to handily TD inventory with the necessary information to support decisions. It was also possible to compare estimated versus realized when fixing TD, and it helps on monitoring individual performance and accurately estimation of TD. With continued development, getting more people involved on technical debt scenario and its management, it is expected to discover ways to further support the projects’ TD management through enhancements in the TD-Manager.

We planed TD-Manager tool to enhance productivity for industrial settings. So, we expect to further improve and validate TD-Manager in such environments. For that, we have planned an extensive series of case studies with an experimental evaluation on open source software. These studies will cope with TD management over finished and ongoing open source software projects with characteristics desirable like access to software documentation, issues repository and existence of test plans. For finished products we intend to use apply technical debt identification and assessment tools in order to simulate the life-span of technical debt.

REFERENCES


Int'l Conf. Software Eng. Research and Practice | SERP'15 |


Figure 4: Implemented Integrated Catalog