Identification and Management of Technical Debt: A Systematic Mapping Study

Nicolli S.R. Alves[2], Thiago S. Mendes[1,4], Manoel G. de Mendonça[1], Rodrigo O. Spínola[1,2], Forrest Shull[5], Carolyn Seaman[3]

[1]Fraunhofer Project Center for Software and Systems Engineering at Federal University of Bahia (UFBA), Salvador, Bahia, Brazil

[2]Graduate Program in Systems and Computer, Salvador University, Salvador, Bahia, Brazil

[3]Department of Information Systems, University of Maryland Baltimore County, Baltimore, MD, USA

[4]Information Technology Department, Federal Institute of Bahia – IFBA, Santo Amaro, Bahia, Brazil

[5]Carnegie Mellon University, Software Engineering Institute, Arlington, VA, USA

nicollirioss@gmail.com, thiagomendes@dcc.ufba.br, manoel.mendonca@ufba.br, rodrigo.spinola@pro.unifacs.br, fjshull@sei.cmu.edu, cseaman@umbc.edu

Abstract

Context: The technical debt metaphor describes the effect of immature artifacts on software maintenance that bring a short-term benefit to the project in terms of increased productivity and lower cost, but that may have to be paid off with interest later. Much research has been performed to propose mechanisms to identify debt and decide the most appropriate moment to pay it off. It is important to investigate the current state of the art in order to provide both researchers and practitioners with information that enables further research activities as well as technical debt management in practice.

Objective: This paper has the following goals: to characterize the types of technical debt, identify indicators that can be used to find technical debt, identify management strategies, understand the maturity level of each proposal, and identify what visualization techniques have been proposed to support technical debt identification and management activities.

Method: A systematic mapping study was performed based on a set of three research questions. In total, 100 studies, dated from 2010 to 2014, were evaluated.

Results: We proposed an initial taxonomy of technical debt types, created a list of indicators that have been proposed to identify technical debt, identified the existing management strategies, and analyzed the current state of art on technical debt, identifying topics where new research efforts can be invested.

Conclusion: The results of this mapping study can help to identify points that still require further investigation in technical debt research.

Keywords: Technical Debt, Software Maintenance, Software Engineering, Systematic Mapping.

1. Introduction

The technical debt (TD) metaphor was first mentioned by Ward Cunningham in 1992 [Cunningham, 1992]. His definition, "not-quite-right code", remains the most commonly cited, but

it has been extended to refer to those internal software development tasks chosen to be delayed, but that run a risk of causing future problems if not done eventually. Thus, it describes the debt that the development team incurs when it opts for an easy or quick approach to implement in the short term, but with a greater possibility of a negative long-term impact.

Debt can refer to any aspect of the software that we know is inappropriate, but do not have time to fix at the moment, such as outdated/missing documentation, planned testing that is not executed, overly complex code that needs to be restructured or refactored, and known defects that remain uncorrected. The result of these immature artifacts is observed in unexpected delays in carrying out necessary modifications, and in difficulties meeting the established quality criteria of the project [Spínola et al., 2013] [Zazworka et al., 2013].

TD is usually incurred in software projects when there is a need to choose between maintaining the quality standards of the system, and putting the software to work in the shortest possible time, using minimal resources. These TD "items", or instances, may have to be paid with interest later in the project. Translating this metaphor into a tractable model for analysis, we identify the following variables:

- The *principal* on the debt refers to the cost to eliminate the debt (i.e. the effort required to complete the task);
- The *interest amount* 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 [Seaman and Guo, 2011], including the extra cost of paying off the debt later, as compared to earlier;
- It is also necessary to consider the *interest probability*, because TD will not always bring negative impacts on future project activities. For example, the higher the probability that the artifact that contains the debt will undergo maintenance, the higher the probability that the interest will negatively impact the project.

To illustrate the aforementioned variables, we can imagine a scenario where a software product, over time, becomes highly coupled and contains many redundant modules. Reducing the coupling and cleaning up the code constitutes the *principal* on this debt. Although the software may be functioning properly, any addition of new functionalities may be time consuming and require extra effort to deal with the coupling or redundancy issues. The probability that extra effort will be required is the *interest probability*, while the amount of extra effort that is likely is the *interest amount*. Although such design decisions do no harm in the current stage, or may even have benefits such as reduced design time, these immature artifacts can be seen as a type of debt that may burden software maintenance in the future.

Despite similarities between terms and concepts that are used, technical debt is not the same as financial debt. The major difference is that the interest associated with technical debt may or may not need to be paid off [Guo et al. 2014]. By incurring technical debt, software managers can trade off software quality against productivity. If on one side maintenance time or cost is reduced in the short term (which is the main advantage of incurring technical debt), on the other side, this advantage is achieved at the cost of extra work in the future [Guo et al. 2014]. Therefore, software managers have to balance the costs and benefits of technical debt and make informed decisions on when and what technical debt should be paid off [Lim et al. 2012].

In order to ensure productivity in the short term and at the same time monitor the progress of the project so that incurred debt doesn't impede the development of the project, TD management techniques have started to be developed [Seaman et al., 2012]. These techniques are generally concerned with identifying and monitoring TD items (instances of technical debt) so that they are explicit and are paid at the right time.

But even before we can effectively work on the management of debt, we need to know what types of debt can be incurred, how they can be identified, and what strategies can be used to manage

it. Although technical debt is being increasingly discussed, as reported by trends.google.com indicating that over the last seven years more and more Google users have been searching for the term "Technical Debt"), it is still difficult to have a broad understanding of the area because the information about it is still spread out in the technical literature.

Beyond a general investigation of technical debt identification and management techniques, we focus in particular on software visualization techniques. Software visualization techniques have been used in software engineering as a possible solution to the task of software systems understanding. Software visualization uses visual aids to facilitate software comprehension [Novais et al., 2013]. While it seems clear that tools that have been found useful for software comprehension should be highly useful in the identification and management of technical debt, it is still not clear how visualization techniques can support TD related activities. Thus, in this study, we specifically examine the literature that suggests ways that this might be done.

In this context, this work presents a systematic mapping of the literature, conducted to address the following high-level research question: "***What are the strategies that have been proposed to identify or manage TD in software projects?***". The following complementing research questions were derived from the main question:

- **(Q1)** What are the types of TD?
- **(Q2)** What are the strategies proposed to identify TD?
  - **(Q2.1)** Which empirical evaluations have been performed?
  - **(Q2.2)** Which artifacts and data sources have been proposed to identify the TD?
  - **(Q2.3)** Which software visualization techniques have been proposed to identify TD?
- **(Q3)** What strategies have been proposed for the management of TD?
  - **(Q3.1)** Which empirical evaluations have been performed?
  - **(Q3.2)** Which software visualization techniques have been proposed to manage TD?

By answering these questions, in this study, we have identified the types of TD, the indicators of their existence in projects, and the strategies that have been developed for the management of this debt. Further, we assess the degree of maturity of the existing proposals through an analysis of the empirical evaluations that have been carried out. In addition, we also investigated how software visualization capabilities have been used to support the identification and management of TD by identifying which visual metaphors have been proposed and what platforms are being used to show the different types of debt. These results contribute to the evolution of the TD Landscape [Izurieta et al., 2012].

We believe that the results of the study presented in this paper will be beneficial for both researchers and practitioners. For the research community, this mapping will provide information about the current status of TD research, as well as topics that require further investigation. For practitioners, the paper shows the types of TD currently considered, as well as strategies for their identification and management. Professionals may use this information as a basis for adapting and developing strategies to control the TD in their projects.

Besides this introduction, this paper has seven other sections. Section 2 discusses some related work. In section 3, the methodology used in this work is presented. Section 4 presents our implementation of the research methodology, including the process of defining the addressed research questions, the study selection process, and the classification scheme we used. Next, in section 5, the results of the systematic mapping are shown. Section 6 discusses the results, compares them to related work, and presents implications for practitioners and researchers. Section 7 presents the threats to the validity of the study. Finally, Section 8 presents the conclusions of this work and directions for further research.

2. Related Work

Technical debt has been increasingly investigated in recent years. An indicator of this trend

is the existence of five other secondary studies in the area. In this section, we will discuss in chronological order the goals and results of each study.

Tom et al. (2012) presented, to the best of our knowledge, the first secondary study in the area. By performing a systematic review, Tom et al. intended to provide a consolidated understanding of the TD phenomenon (research questions: *What are the elements of technical debt? Why does technical debt arise?*), to reflect this consolidated understanding in the form of a theoretical framework, and discuss the positive and negative outcomes of TD (research question: *What are the benefits and drawbacks of allowing technical debt to accrue?).* According to the authors, the resulting theoretical framework portrayed a holistic view of TD that incorporates a set of precedents and outcomes, as well as the phenomenon itself (behaviors, metaphors, and elements).

In another secondary study in the area, Villar and Matalonga (2013) performed an initial mapping of the area by answering the following research questions:

- Which are the current definitions of technical debt and development debt?
- What research activities have been performed in the area?
- How has the area evolved over time?
- Who are the main researchers in the area?

As results, the authors presented:

- 11 definitions of TD;
- the number of published papers over the years;
- four categories (general papers on TD, code debt, other types of debt, stakeholders that deal with TD in their projects) that were created to group the analyzed papers, and;
- a list of the most active (in terms of published papers) authors in the area.

Tom et al. (2013) extended the work from Tom et al. (2012) by performing an exploratory case study that involved a multivocal literature review, using accessible writings such as internet blogs, white papers, and trade journal articles, supplemented by interviews with software practitioners and academics to establish the boundaries of the TD phenomenon. The research goal of this study was to consolidate understanding of the nature of technical debt and its implications for software development, thus establishing the boundaries of the phenomenon and a more complete theoretical framework to facilitate future research. The findings of this study included the creation of a useful theoretical framework, consisting of a set of TD dimensions, attributes, precedents and outcomes, as well as the phenomenon itself and a taxonomy that describes and encompasses different forms of TD. However, it does not provide a comprehensive taxonomy either of the types of TD, nor of indicators that can be used to support the identification of different types, that is commonly accepted and broadly used by the research community. Further, this study did not focus on the research literature in a way that allows future researchers to build on existing work.

More recently, Ampatzoglou et al. (2015) performed a systematic review focusing on the financial perspective in the discussion about TD. The goal of their study was to analyze research efforts on technical debt, by focusing on their financial aspect. Specifically, the analysis was carried out with respect to how financial terms are defined in the context of technical debt and how they relate to the underlying software engineering concepts. The authors found that the most common financial terms that are used in technical debt research are principal and interest, whereas the financial strategies that have been more frequently applied for managing technical debt are real options, portfolio management, cost/benefit analysis and value-based analysis. The authors also emphasized that the application of such strategies lacks consistency, i.e., the same strategy is differently applied in different studies, and in some cases lacks a clear mapping between financial and software engineering concepts.

In another significant related work in this area, Li et al. (2015) performed a systematic

mapping study aimed at collecting studies on TD and TD management, and performing a classification and thematic analysis on these studies. Their main goals were to get a comprehensive understanding of the concept of TD, an overview of the current state of the research on TD management, identify the quality attributes that are compromised when TD is incurred, and promising future research directions. As a result, TD was classified into 10 types, 8 TD management activities (for instance: TD identification, repayment, prevention, communication, and monitoring) were identified, and 29 tools for TD management were collected.

As can be seen, other studies shared research questions Q1 and Q3, but took slightly different perspectives in terms of scope and how collected data was categorized. A discussion on the differences will be presented in section 6.1.

Finally, it should be noted that the previously published preliminary results of this study [Alves *et. al*, 2014] consist of a taxonomy of types of TD represented as a lightweight ontology. In this paper, we also summarize this ontology, and present the rest of the results of the mapping study.

3. Mapping Study Method

Systematic literature mapping is a useful tool for achieving quality of information in a literature review. It provides a means to perform comprehensive and unbiased literature reviews, providing considerable scientific value. The systematic mapping (SM) is a type of secondary study that aims to characterize a particular area of research through a systematic procedure whose purpose is to identify the extent and nature of the primary studies available in the area [Budgen et al., 2008].

While a systematic review (another controlled approach to conducting secondary studies) is a way of identifying, evaluating and interpreting all available relevant research on a particular issue [Kitchenham et al., 2007], the SM intends to "map" the investigation instead of answering the research question in detail. Budgen et al. (2008) states that the early stages of a mapping study are generally very similar to those of a systematic literature review, although the research question itself is likely to be much broader, in order to adequately address the wider scope of such a study. In this work, we choose to perform systematic mapping rather than systematic review because of the wider scope of our study. We intend to search the literature to determine what sorts of studies addressing the research question on TD identification and management have been carried out, where they are published, in what databases they have been indexed, and what sorts of outcomes they have assessed. Both data extraction and analysis are largely concerned with classification of the available studies.

A well-organized set of guidelines and procedures for carrying out SMs in the context of Software Engineering is defined [Petersen et al., 2008] [Budgen et al., 2008], which lays the foundation for the study presented in this paper. The main reasons to perform a SM are to systematically identify gaps in the current body of research and support the planning of new research, avoiding the unnecessary duplication of effort and error [Budgen et al., 2008]. It is worth noting that the importance and use of SM in the area of Software Engineering is growing [Petersen et al., 2008] [Budgen et al., 2008], showing the relevance and the potential of the method.

The empirical software engineering research community has defined a standard process for conducting this type of study [Peterson et al., 2008]. Figure 1 shows the phases of SM used in this study. The execution of each phase will be explained in detail in the following sections.

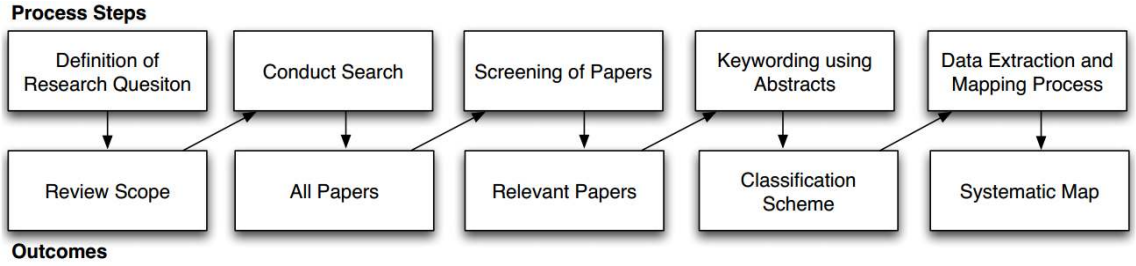

Figure 1. The systematic mapping process (adapted from [Petersen et al.**, 2008]).**

4. Study Implementation

In this section, we explain how we implemented the process outlined in Figure 1, including our research questions, our specific search strategy and terms, our selection criteria and process, and our classification scheme.

4.1 Definition of Research Questions

For this study, a primary research question was defined: "**What are the strategies that have been proposed to identify or manage TD in software projects?**". The following complementary research questions were derived from the main question. By answering these questions, we will have a detailed characterization of the identified studies:

Q1. What are the types of TD?

Many initiatives have been undertaken to investigate the TD research area from different perspectives, e.g. what causes TD, what effects TD has, how to avoid TD, etc. One perspective involves the different types of TD that can be found in software projects. It is necessary to organize this knowledge about existing types so that the TD research community can share a common vocabulary. Thus, this research question aims to identify the different types of TD found in the study.

Q2. What are the strategies proposed to identify TD?

Once the types of TD are known, it is also important to know how they can be identified and documented in software projects. There are several ways of identifying TD of different types. Some of them involve simply making explicit (and in some cases quantifying) TD that is not really hidden, e.g. outdated documentation or incomplete tests. Other identification strategies involve the use of tools to analyze source code or other artifacts to find hidden debt, e.g. poorly structured code, architecture problems, etc. Each identification strategy has associated with it some kind of "indicator", sometimes a metric or sometimes something less formal, that can be used to point to areas with specific types of debt. This question aims, for each identified type of TD, to identify the indicators that have been proposed for their identification. This association between types of debt and their respective indicators is important because it will allow, when choosing to manage a particular type of TD, informed decisions about the indicators that could be used.

Q2.1. Which empirical evaluations have been performed?

Beyond proposing technologies for identifying TD, it is important to evaluate technologies that have been developed so that their effectiveness is characterized. Different types of assessments can be used, such as survey, case study and controlled experiment.

In this context, the purpose of this question is to identify what types of empirical studies have been used to assess strategies for the identification of TD proposed in the literature. Obtaining this information is important to get an idea of the level of the maturity of the existing proposals.

Q2.2. Which artifacts and data sources have been proposed to identify the TD?

TD of different types can be identified in different artifacts (e.g. requirements, source code) generated during software development. In addition, various types of data sources (where the artifacts are organized) may contain relevant information that makes the discovery of TD possible. Version control and defect management systems are examples of possible data sources that can be analyzed to discover the debt in a project. The purpose of this question is to know which artifacts and data sources have been proposed to identify TD.

Q2.3. Which software visualization techniques have been proposed to identify TD?

Software is becoming increasingly complex in terms of innovation and size. A consequence of this is the increasing amount of information generated from software development activities. This further complicates the task of analyzing the project artifacts, looking for TD items and also, once identified, monitoring their behavior during the evolution of the software. One of the tools that can be used to make this task easier is software visualization. Visualization techniques allow the representation of information that is often difficult to analyze in textual or tabular form. Considering this scenario, this research question seeks to identify whether, and which, visualization techniques have been proposed in the identification of TD.

Q3. What strategies have been proposed for the management of TD?

Just as important as identifying the TD items in a project, is to implement an efficient and effective management strategy for them. Techniques such as Cost-Benefit Analysis [Seaman and Guo, 2011] and Modern Portfolio Theory [Guo and Seaman, 2011] have been proposed for this purpose. However, various other strategies have also been defined. The purpose of this question is to perform a characterization of the strategies that have been proposed for managing TD. Addressing this question will help us understand, for example, what criteria have been used in the decision to pay the debt or maintain it in the project.

Q3.1. Which empirical evaluations have been performed?

The purpose of this question is to identify what types of empirical studies have been used to assess strategies for the management of TD proposed in the literature.

Q3.2. Which software visualization techniques have been proposed to manage TD?

This research question seeks to identify whether, and which, visualization techniques have been proposed in the management of TD.

4.2 Search strategy

We first chose the following keywords to perform the search in the digital libraries:

- **Population**:
  - *Technical Debt;*
  - *Software Project, Software;*
- **Intervention:**
  - *Practice, technique, method, process;*
  - *Identification, identify, gathering, detection, discovery;*
  - *Management, Monitoring.*

Table 1 presents the search string derived from these keywords.

Table 1. Search String.

(("technical Debt") AND (“Software Project” OR “Software”)) **AND**
(“Practice” OR “Technique” OR “Method” OR “Process”)

AND
(“Identification” OR “Identify” OR “Gathering” OR “Detection” OR “Discovery”) **OR** (“Management” OR “Monitoring”)

However, after some tests performed in digital libraries, it was observed that the search string was too restrictive and returned a small number of papers. This could result in an incomplete mapping process. Therefore, we chose a more general search strategy, using only the following keywords:

- **Population:**
  - *Technical Debt;*
  - *Software.*

For these keywords, the search string shown in Table 2 was defined. We applied this search string to Titles and Abstracts. We chose not to do full text search because we found that full text search resulted in a very large number of studies from domains other than software development.

Table 2. Generic search String.

(“Technical Debt”) **AND** (“Software”)

4.3. Data Sources

In choosing data sources, we aimed to include important journals and conferences regarding the research topic. We restricted the search to studies published up to December 2014. We included publications retrieved from several digital libraries and web search engines: ACM Digital Library, IEEE Xplorer, Science Direct, Engineering Village, Springer Link, Scopus, Citeseer, and DBLP. According to [Brereton et al. 2007], these are the recommended data sources of papers for software engineering researchers as they provide access to important journals and conferences in the area.

4.4. Study selection

As the search string was generic, the search returned many papers that were not relevant for this research. Thus, it was necessary to filter out the irrelevant papers, which required a set of criteria.

Our inclusion criteria were:

- Published papers that describe how to identify and/or manage TD;
- When several papers reported the same study, only the most recent was included;
- When multiple studies were reported in the same paper, each relevant study was considered separately.

The exclusion criteria were:

- Papers that do not have information on how they handle the identification or management of TD;
- Papers that are only available in the form of workshop/conference reports, abstracts or Power Point presentations;
- Duplicate papers.

The identification and filtering of the papers was divided into five stages, as shown in Figure 2. The first step consisted of searching for papers using the search string in each of the digital libraries selected for this study (ACM Digital Library, IEEE Xplore, Science Direct, Engineering Village, Springer Link, Scopus, Citeseer and DBLP).

In the second step, the first filtering (Figure 2 - **Filter1**) took place and was performed by only one researcher. In this process, the exclusion criteria were used, removing all the proceedings abstracts, Power Point presentations, and duplicate papers.

In the third step a second filter (Figure 2 - **Filter2**) was applied. Each paper was analyzed by two researchers, and in the event of a conflict, a third researcher analyzed them and took the most appropriate decision. Filtering was carried out by reading the titles, abstracts and introductions (if necessary) using the inclusion and exclusion criteria. When a decision was not yet possible, the paper proceeded to the next step, where it was read in full. After this step, 100 papers were selected to go on to the next filtering stage.

In the fourth stage, the last filter (Figure 2 - **Filter3**) was applied. This time, the selected papers were read in full. At the end of this stage, 12 more papers were excluded. For example, Cai et al. (2013) reported on an experiment applying a tool-supported architecture review into software design education and Tomas et al. (2013) has done a study on the existing tools to calculate metrics of internal quality on software projects. However, the focus of these studies was different from the research question and/or did not have enough information about the identification or management of TD specifically.

Finally, in the fifth stage we applied the snowballing by checking the references of each selected study (89) in order not to miss any potentially relevant studies (Budgen et al., 2008). At the end, the selected studies (11) from the snowballing process were combined into the final results of the study selection. The remaining 100 (listed in Appendix A) papers were used to extract the information to answer the research questions.

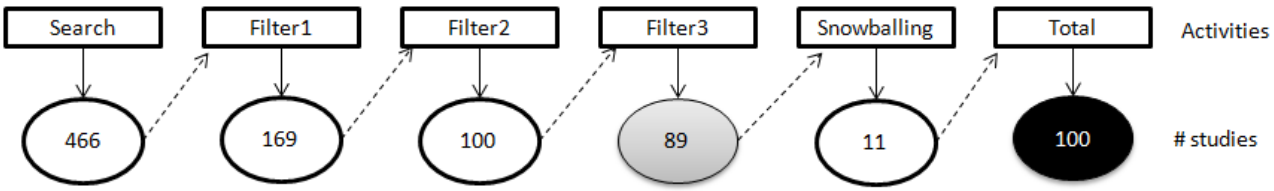

Figure 2. The filtering process of the papers

The final number of selected papers from each digital library is shown in Figure 3. Clearly, most of the selected papers came from ACM Digital Library and IEEE Xplore online libraries.

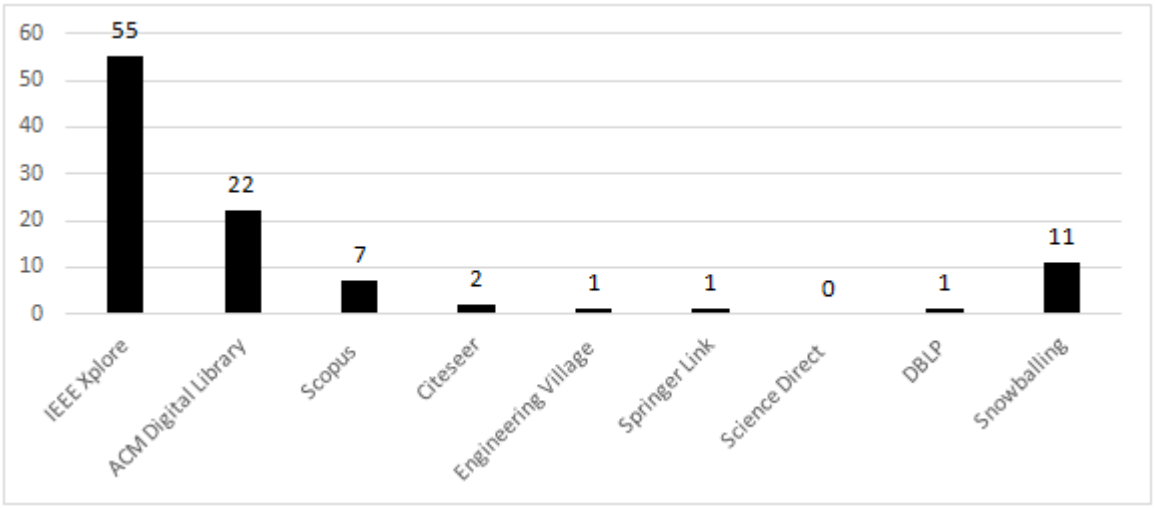

Figure 3. Number of papers included by digital library.

4.5 Classification scheme

The attributes in the classification scheme were structured into six categories to allow a better information analysis: metadata about the selected studies, TD types, TD indicators, management strategies, evaluation studies, and strategies for software visualization. Each category is related to one or more research questions (as defined in section 3) and is presented below.

− Metadata of the studies

The first category contains data about the selected studies such as: venues where the studies were published, authors and affiliations of the papers, type of the papers (e.g. short, full or journal papers), and year of publication.

To collect the information for this category, we counted the number of papers per publication venue, as well as the type of venue (e.g. conference, workshop, or journal) where they were published. In addition, the papers were classified into two types: short papers and full papers. Papers with up to 4 pages are considered short papers; longer papers are full papers.

− TD types (Q1)

This category includes information about the TD types. Basically, its attributes are TD types and their definitions. 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. Different types of TD have differing nature depending on the moment in which they are incurred or on the activities that they are associated with. Multiple types of TD have been studied and presented in the literature.

The types of debt are usually identified in papers in one of the following ways:

- Direct: name of the type + the word debt (for instance: design debt, defect debt) or;
- Indirect: indentified in phrases such as *the developer incurred a type of debt related to the architecture of the project* (architecture debt) or *this type of debt is derived from late decisions regarding training people or hiring developers* (people debt).

Thus, in order to identify the different types of debt in each paper, three researchers documented the types and their definitions from each paper, using terminology straight from the papers. At the end, this information was consolidated for each type of debt that was found. The types of debt are not orthogonal, i.e. some instances of TD could conceivably fit into more than one category.

− TD indicators (Q2, Q2.1, and Q2.2)

In the third category, information about the identified TD indicators is represented. Basically, its attributes are the list of indicators, their relationship with each TD type, the type of empirical evaluation performed on each indicator, and where (data source) each indicator can be found in software projects.

In order to identify different TD indicators, during the reading of the selected studies, three researchers collected the mentioned indicators as well as types of debt they are associated with and software development artifact in which they are identified, following the terminology straight from the papers. At the end, this information was consolidated.

− TD management strategies (Q3 and Q3.1)

The fourth category represents the TD management strategies. We listed all strategies described in the literature with the aim to manage any type of TD in the project. Besides the strategies and their definitions, we also investigated the type of empirical evaluation performed on them.

In order to identify the different strategies, three researchers collected the mentioned strategies and their corresponding definitions following the terminology straight from the papers. To be considered a management strategy, we adopted the criteria that the strategy needs to support decisions about when and if a TD item should be paid. At the end, this information was consolidated for each management strategy that was found.

− TD evaluation studies (Q2.1 and Q3.1)

Software engineering aims to support the development of software systems within previously established limits of time, cost and quality [Pfleeger, 2007]. However, according to Kitchenham et al. (2004), using good processes is not enough to improve quality during software development. This is because development is dependent on technologies that do not often present sufficient evidence of their potential benefits, limitations, cost of implementation and associated risks. To deal with this, Kitchenham et al. (2004) argue that the use of evidence would allow the characterization of a technology prior to its adoption in software projects by the industry, so that it would be possible to determine, with reasonable levels of confidence, the feasibility of its use when considering specific use scenarios.

In this context, there are different types of empirical studies that could be used to gain evidence about the feasibility and effectiveness of any proposed strategy. The application of these types of study, and consequently the use of the empirical paradigm, to support evaluation in software engineering is important because they contribute to a higher level of maturity. This also applies to research on TD.

The fifth category shows the collected information about empirical studies found in the literature. To classify the types of performed evaluations, we used the following taxonomy [Wohlin et al., 2000]:

- **Case study**: used to monitor projects, activities or assignments aiming to trace a specific attribute or establish relationships between different attributes without much formal control over the activities related to the experimental method;
- **Controlled experiment**: an empirical enquiry that manipulates one factor or variable of the studied setting. Based on randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and the effects on outcome variables are measured. In human-oriented experiments, humans apply different treatments to objects, while in technology-oriented experiments, different technical treatments are applied to different objects;
- **Ethnographic study**: applied to understand user behavior in detail. This is a specialized type of case study with focus on cultural practices or long duration studies with large amounts of participant-observer data.

− TD software visualization techniques (Q2.3 and Q3.2)

Software visualization techniques have been investigated in software engineering to help in understanding, maintaining, testing, and evolving software systems [Novais et al., 2013]. Software visualization techniques are increasingly researched, motivated by the fact that vision is the most used sense by humans [Diehl, 2007]. In this sense, software visualization techniques can support the developer in the identification and/or management of different types of TD in software projects. Thus, it is important to investigate which software visualization techniques have been proposed to support the identification and/or management of TD and what platforms are being proposed to show such visualization techniques.

The sixth category reflects the papers that make use of software visualization techniques to identify and manage TD in software projects. We classified the software visualization techniques according to [Novais et al., 2013].

Finally, for analysis of the TD indicators and TD evaluation studies, the papers were classified into two types: papers that analyze (those that present indicators and describe them in detail) and papers that only cite (those that do not go into details).

5. Mapping Results

For data extraction, the included studies were read in full by three researchers who were directly involved in the mapping. A spreadsheet[1] was used to collect and to analyze the data.

In this section we present the analysis of the data extracted from the selected studies. First we present the analysis of metadata related to publications and authors. In subsequent subsections, we present the types of TD, their indicators, management strategies, identified evaluation studies and the types of software visualization strategies being used.

The largest number (38 out of 100) of papers is concentrated in the International Workshop on Managing TD. Another small cluster of papers (5) were found from a special issue of IEEE Software on TD. However, publications are already emerging in other venues, such as the Agile Conference, the IEEE Annual Computer Software and Applications Conference, the International Conference on Software Engineering (ICSE), the IEEE International Conference on Software Maintenance (ICSM), the International Symposium on Empirical Software Engineering and Measurement (ESEM), the Cutter IT Journal, and the Software Quality Journal. There were a total of 51 different venues with 1 or 2 papers each.

As can be seen in Figure 4, the study distribution over publication types is 40% (40 studies) for workshops, 32% (31 studies) for conferences, 4% (4 studies) for symposia, 15% (15 studies) for journals, and 9% (9 studies) for magazines. If we merge the publication types conference and symposium, and journal and magazine (peer-reviewed) into two groups, we will have a distribution of studies close to that obtained by Li et al. (2015). Besides, there has been an increase in the number of both full papers and publications in journals, which indicates that the area of research is maturing.

[1] https://drive.google.co]m/file/d/0B_x3JCtCrSomdW56NFJqQnJ2N00

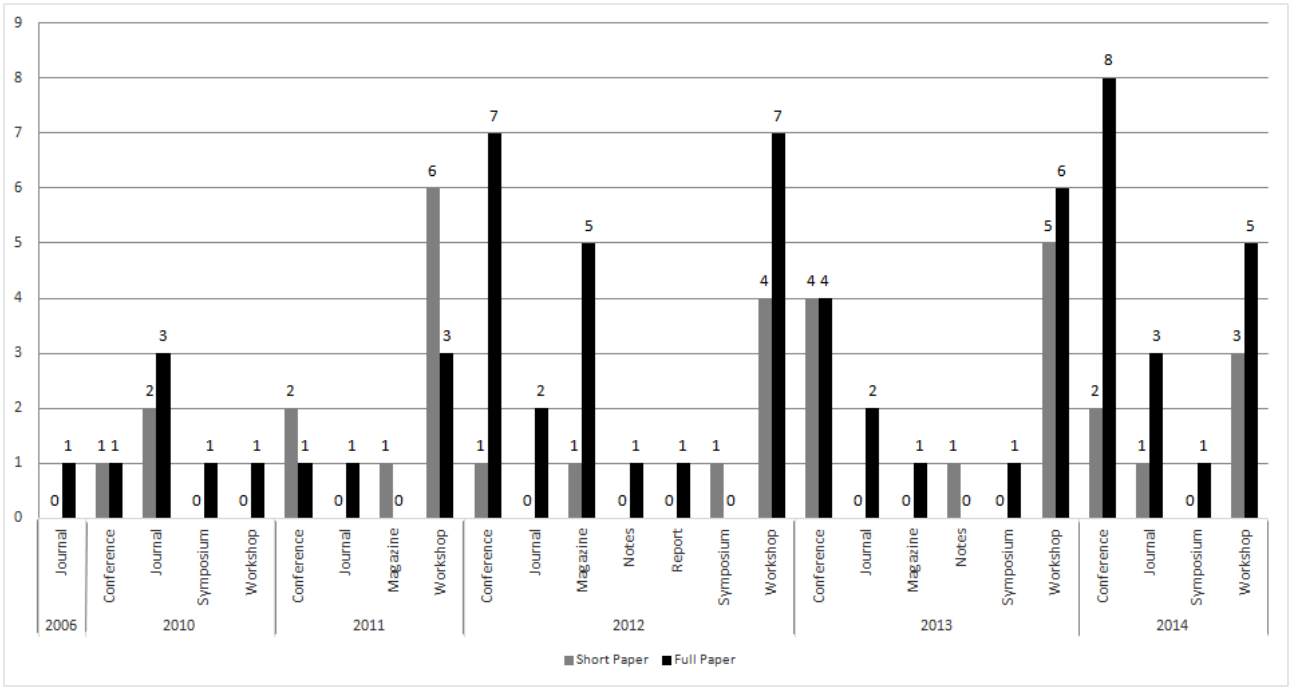

Figure 4. Studies by manner of publication, type of paper and by year.

The 100 selected papers were written by 115 different authors, showing a wide interest on this subject in the software engineering research community. However, it was found that only 10 researchers (Antonio Vetro', Carolyn Seaman, Clemente Izurieta, Forrest Shull, Ipek Ozkaya, Nico Zazworka, Rami Bahsoon, Robert Nord, Yuanfang Cai and Yuepu Guo) were involved in more than 5 papers each. Of this group, the five most active authors are all very close collaborators (22 of the 100 papers had at least one of these authors).

5.1. Technical debt types (Q1)

Here, the types found in this mapping study are presented sorted by frequency, as well as their definitions (an initial version of this taxonomy of types, based on selected papers up to 2013, is reported separately [Alves *et. al*, 2014]):

- **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) [Guo and Seaman, 2011] [Izurieta et al., 2012];
- **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 [Brown et al., 2010] [Kruchten et al., 2012];
- **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 [Guo and Seaman, 2011];
- **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) [Guo and Seaman, 2011];
- **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 [Bohnet and Dцllner, 2011];
- **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 [Snipes et al., 2012];

- **Requirements Debt:** Refers to tradeoffs made with respect to what requirements the development team needs 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.) [Kruchten et al., 2012];
- **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 [Seaman and Spínola, 2013];
- **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 [Seaman and Spínola, 2013];
- **Test Automation Debt**: Refers to the work involved in automating tests of previously developed functionality to support continuous integration and faster development cycles [Codabux and Williams, 2013]. This debt can be considered a subtype of test debt;
- **Process Debt**: Refers to inefficient processes, e.g. what the process was designed to handle may be no longer appropriate [Codabux and Williams, 2013];
- **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 [Morgenthaler et al., 2012];
- **Service Debt:** Refers to the inappropriate selection and substitution of web services that lead to mismatch of the service features and applications' requirements. Besides, this type of debt also leads to under- or overutilization of the system by integrating a service that does not use the system resources in the expected way (for instance, lack of memory due to a service that does not follow the expected data processing process, or lack of performance due to a service that does not use the available memory for the task). This kind of debt is relevant for systems with service-oriented architectures [Alzaghoul and Bahsoon, 2013];
- **Usability Debt**: Refers to inappropriate usability decisions that will need to be adjusted later. Examples of this debt are lack of usability standard and inconsistence among navigational aspects of the software [Zazworka et al., 2013][Potdar and Shihab, 2014];
- **Versioning Debt**: Refers to problems in source code versioning, such as unnecessary code forks [Greening, 2013].

Table 3 shows the number of papers that analyzed or mentioned each TD type identified in this mapping. It is also observed that some papers did not discuss a specific type of debt, and therefore are represented on the graph by the term "Technical Debt".

Table 3. Papers by type of TD over the years.

TD Type	2006	2010	2011	2012	2013	2014	Total
Design	1	5	8	11	9	8	**42**
Architecture	0	2	3	11	5	9	**30**

Documentation	0	2	4	6	4	12	**28**
Test	0	2	2	8	6	6	**24**
(Type not specified) Technical Debt	0	1	1	5	6	10	**23**
Code	0	3	1	9	5	3	**21**
Defect	0	1	5	3	3	5	**17**
Requirement	0	0	0	2	0	2	**4**
Infrastructure	0	1	0	1	1	0	**3**
People	0	0	0	1	0	2	**3**
Test Automation	0	0	0	0	2	1	**3**
Process	0	0	0	0	2	1	**3**
Build	0	0	0	1	0	1	**2**
Service	0	0	0	0	2	0	**2**
Usability	0	0	0	0	1	1	**2**
Versioning	0	0	0	0	1	0	**1**

Table 3 also shows that, in the years 2010 and 2011, papers were highly concentrated on architecture, design and documentation debt, along with some papers on code and test debt. Moreover, it can be seen that the TD types have expanded with time, indicating that new fields are being included, such as service, process, usability, and versioning. We can also observe that there is a high concentration of studies on types of debt more related to the source code (design, architecture, code, and defect). A possible explanation for this is that there is a plethora of tools that perform source code analysis and can be used to support the detection of TD from the source code.

5.2. Technical Debt Indicators (Q2, Q2.1, Q2.2, Q2.3)

TD indicators allow the discovery of TD items when analyzing the different artifacts created during the development of a software project. Table 4 shows the indicators that were identified in this study organized by the TD type that they are associated with. To map indicators to types of TD, we used information provided by the papers where each indicator was identified. It is also important to mention that the identified indicators were explicitly mentioned in the papers as a way to identify a specific type of debt, for example: *god classes were used to support the identification of design debt* or *one alternative to identify architectural issues is to look for modularity violations.*

We can observe that some types, such as design, already have a fair number of indicators. On the other hand, indicators were not identified for some types of debt in the literature: process, infrastructure, people, and usability debt. We also observe that, just as the types of debt are not orthogonal, we also have some indicators that are mapped to more than one type. 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 overlap between these TD types and, as a consequence, we also have some overlap between their indicators. Besides, it is also possible to have indicators that can be extracted from the code but that are related to other types of debt, other than code debt. For example, indicators of design debt are extracted from the source code, but some of them cannot be considered indicators of code debt because they reflect the lack of object oriented design practices.

Table 4 also shows the number of papers that identified each indicator. For this analysis, the papers were classified into two types: papers that analyze (those that present indicators and describe them in detail) and papers that only cite (those that do not go into details). It can be observed that, with some exceptions (e.g. Code Smells, Automatic Static Analysis - ASA Issues, Documentation, Coding Standards, and Modularity Violation), there are few papers detailing each indicator. In fact, most of the identified indicators are either cited or analyzed by only one paper. This indicates that more studies need to be performed in order to investigate the real benefits and limitations of each indicator when identifying items of TD in software projects.

Table 4. Indicators organized by TD type

Indicators	Articles that analyze	Articles that only cite	TD Type
Violation of Modularity	***6***	*3*	*Architecture Debt*
Software Architecture Issues	***7***	*2*	
Betweenness Centrality	***1***	-	
Augmented Constraint Network (CAN)	***1***	-	
Pairwise-Dependency Relation (PWDR)	***1***	-	
Index of Package Changing Impact (IPCI)	***1***	-	
Index of Package Goal Focus (IPGF)	***1***	-	
Structural Dependencies	***3***	-	*Architecture Debt / Build Debt*
Structural Analysis	***1***	*1*	*Architecture Debt / Design Debt*
Build Issues	***3***	-	*Build Debt*
Code without Standards	***3***	*5*	*Code Debt*
Slow Algorithm	-	*1*	
Multithread Correctness	***1***	*1*	
Code Metrics (not specified)	***3***	*3*	*Design Debt / Code Debt*
Automatic Static Analysis (ASA) Issues	***7***	*2*	
Code Smells	***38***	*14*	
Grime	***3***	*2*	*Design Debt*
Software Design Issues	***2***	*2*	
Low External / Internal Quality	***1***	*2*	
Afferent / Efferent Couplings (AC / EC)	***1***	-	
Depth of Inheritance Tree (DIT)	***1***	-	
Referential Integrity Constraints (RICs)	***1***	-	
Uncorrected Known Defects	***3***	*3*	*Defect Debt / Test Debt*
Insufficient Comments in Code	-	*1*	*Documentation Debt*
Lack of Documentation	***1***	-	
Comments (hack, fixme, is problematic, ...)	***1***	-	
Documentation Issues	***6***	*11*	
-	-	-	*Infrastructure Debt*
-	-	-	*People Debt*
-	-	-	*Process Debt*
Requirement Backlog List	***1***	-	*Requirement Debt*
Selection/Replacement of Web Service	***1***	-	*Service Debt*
Lack of Automated Testing	***1***	-	*Test Automation Debt*
Incomplete Tests	***3***	*4*	*Test Debt*
Defects Deferred	***3***	-	

Insufficient Code Coverage	***1***	-	
Lack of Test Case Documentation	***1***	-	
Lack Test Case planning	***1***	-	
-	**-**	-	*Usability Debt*
Unnecessary code forks	-	1	*Versioning Debt*

The results show that the most cited and analyzed TD indicator is Code Smell. Figure 5 shows the detail of the types of code smells that have been considered as indicators. Some authors did not specify the type of code smell, in these cases we grouped them as "Type not specified". We can observe that the type of code smell that has been most investigated is God Class. An explanation for this is that God Classes are conceptually easy to understand, are up to 13 times more likely to be affected by defects and up to seven times more change-prone than their non-smelly counterparts, what makes them a good candidate when starting to detect TD from the source code [Olbrich et al., 2010][Zazworka et al., 2011].

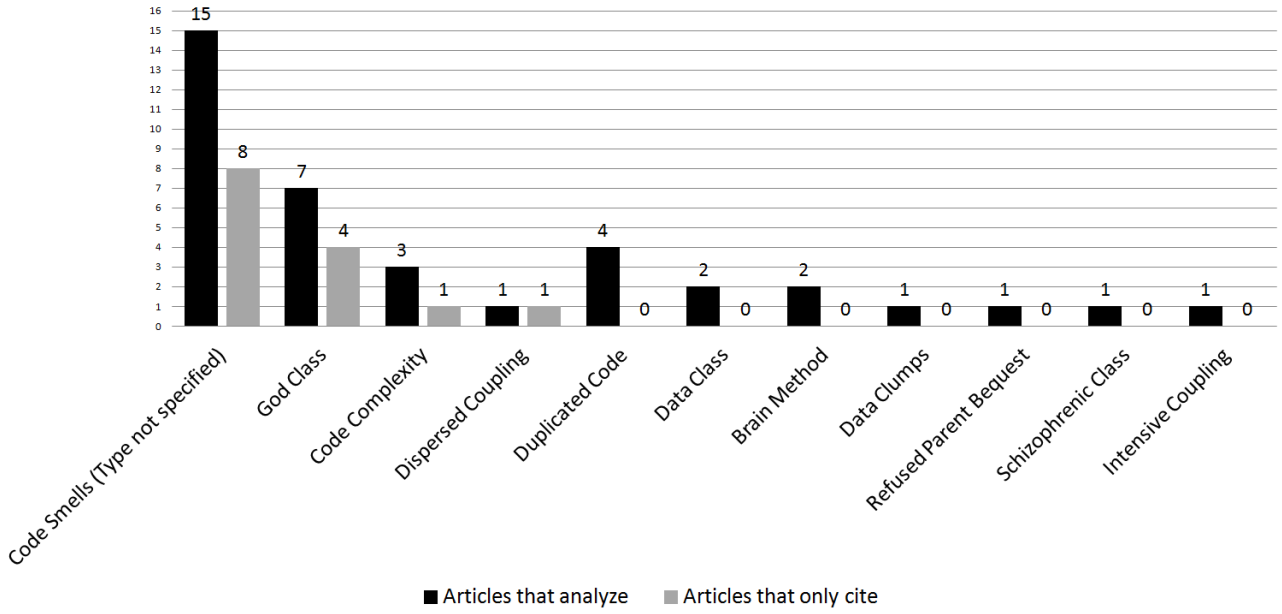

Figure 5. Papers per Code Smell.

5.2.1 Evaluation Studies

The research community has published work on 45 different TD indicators over the years we studied. Four TD indicators appeared in 2010, 7 in 2011, 15 in 2012, 8 in 2013, and 11 in 2014. We also observed that, as the new types of debt appear, new indicators also emerged. Although many indicators have been characterized in this study, we found that few of them have actually been evaluated, i.e. had some empirical evidence assessing their usefulness or validity. To classify the types of evaluation performed, we considered the following taxonomy from Wohlin et al. [2000]: Case study, Controlled experiment, and Ethnographic study.

Our results show that case studies and controlled experiments are the most commonly used types of study among the evaluated indicators. Figure 6 shows the distribution of the number of papers that used each type of empirical study. The case study method was the most used type, and only four controlled experiments were identified. This implies that most of the proposals in the TD area still require more experiments, so that their benefits and limitations can be known with

increased confidence.

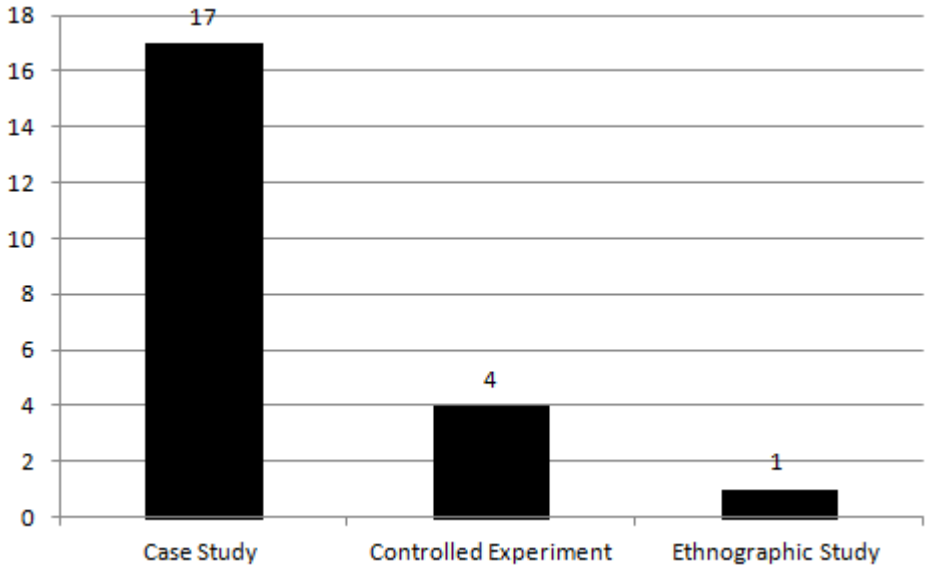

Figure 6. Papers by type of study.

Among the indicators evaluated, we can highlight god class, reported by six studies (three controlled experiments and three case studies). This suggests that the potential usefulness of god class in supporting the activities of TD identification in software projects is already established. Another indicator that stands out is ASA issues, which were evaluated through five case studies and two controlled experiments. Furthermore, Documentation and Software Architecture Issues have been evaluated in five case studies. Violation of Modularity and Structural Dependency indicators have been used in four case studies.

5.2.2 Artifacts and Data Sources

Indicators can also be classified by the software development artifact in which they are identified. Figure 7 shows the artifacts that have most often been used for analysis in the TD literature. There is a clear focus on strategies for identifying TD items from the source code. The other artifacts have only been used occasionally. Again, a possible explanation for this is that there is a set of tools that perform source code analysis and can be used to support the detection of TD from the source code and that there is a high concentration of studies on types of debt more related to the source code.

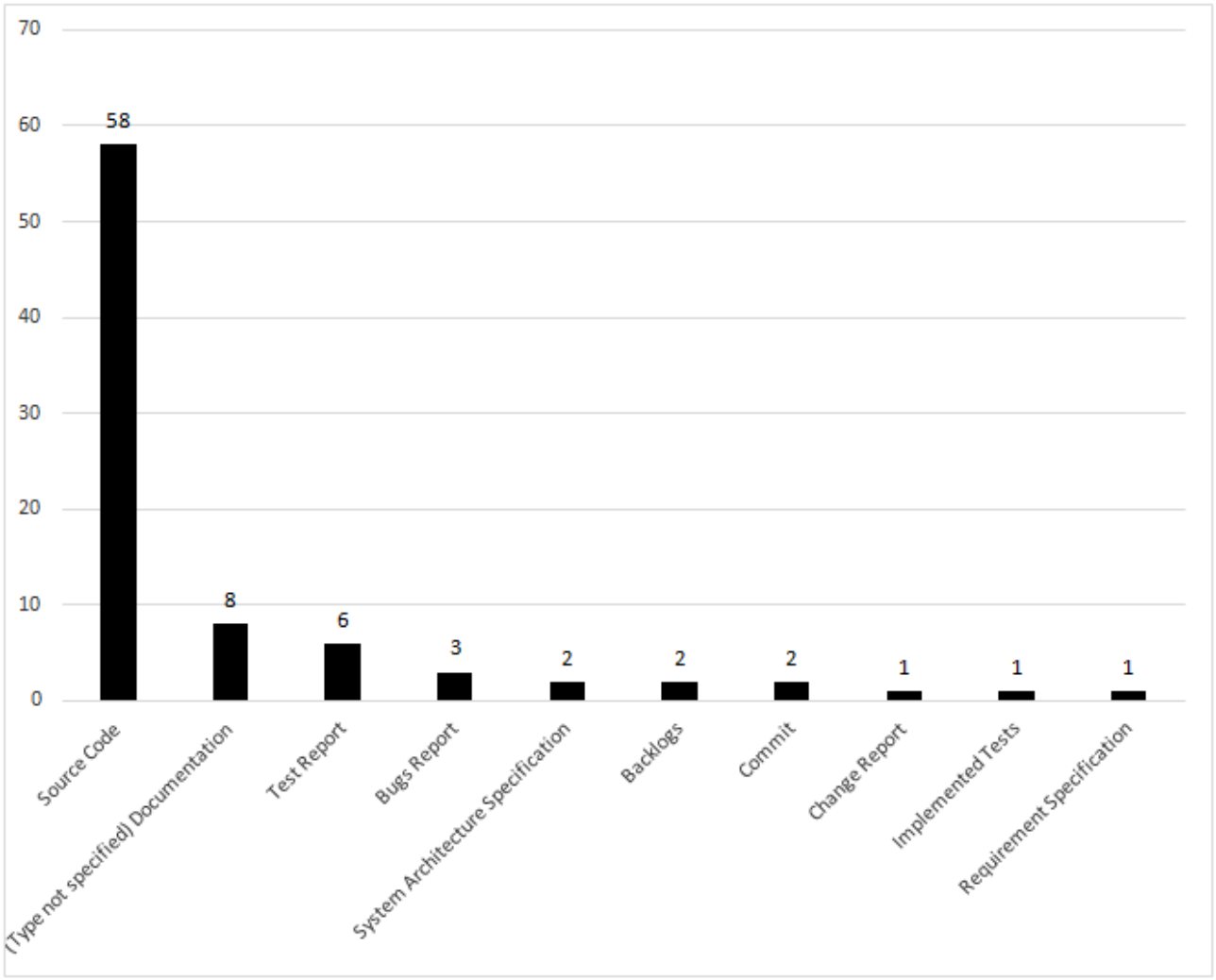

Figure 7. Papers by artifact considered.

As a complement to the results shown in Figure 7, Figure 8 shows the different programming languages used where TD indicators have been investigated in the source code. It should be noted that although different programming languages have been considered, there is a higher concentration on Java. A possible explanation for the frequency of Java-based case studies is the plethora of tools that perform source code analysis in Java.

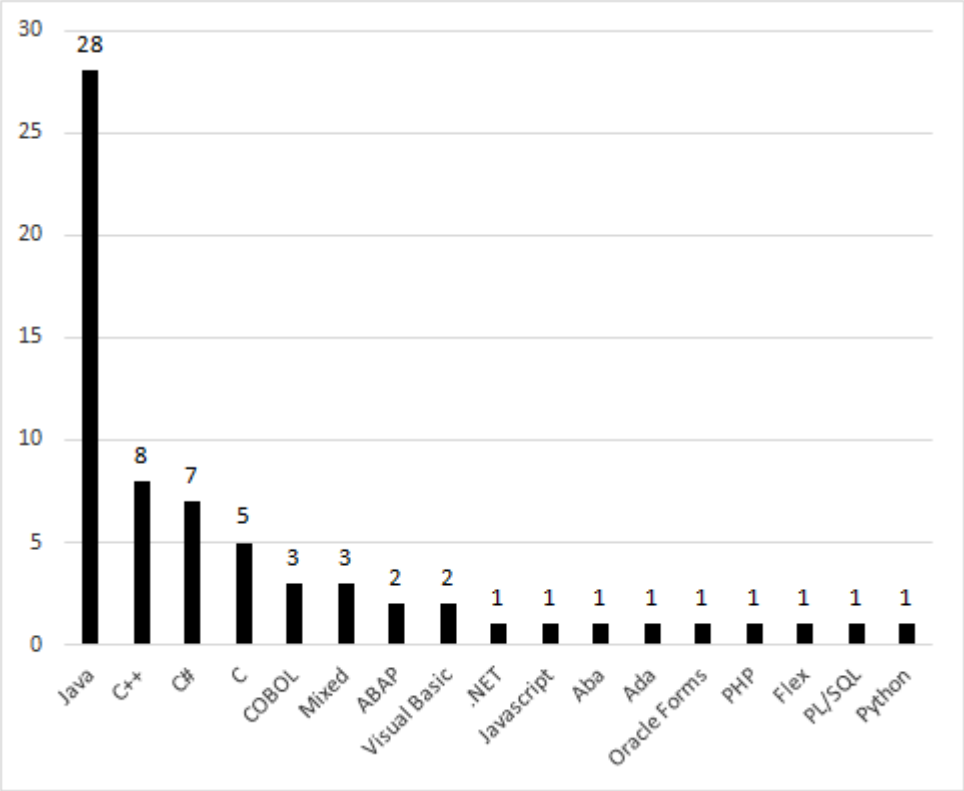

Figure 8. Papers by programming language

Finally, Figure 9 shows the data sources (DS) that have been reported in the literature as sources of useful information in the strategy of identifying TD. For example, if the TD is located in the code, the DS refers to the location where the code is stored. The identified DSs were:

- **Bug Tracking Systems (BTS)**: a software application that keeps track of reported software bugs and records maintenance and change occurrences over the software lifecycle (e.g. Bugzilla[2]);
- **Configuration Management Systems (CMS)**: software that enables the project team to work in a controlled and organized way on the artifacts created during software development (e.g. SVN[3] Systems and GIT[4]);
- **Repositories of Applications**: refers to open databases containing information about products that can be used, free of charge, for collecting metrics and conducting studies (e.g. Google Repository[5]);

The most common data source type used was Repositories of Applications and CMS, in six papers. Repositories of applications are important when conducting studies, however the practical impact of such studies is questionable given that the applications are very different from those maintained by practitioners in the software industry. On the other hand, it was observed that BTSs are also being used in some studies.

[2] www.bugzilla.org
[3] https://subversion.apache.org/
[4] http://git-scm.com/
[5] https://code.google.com/

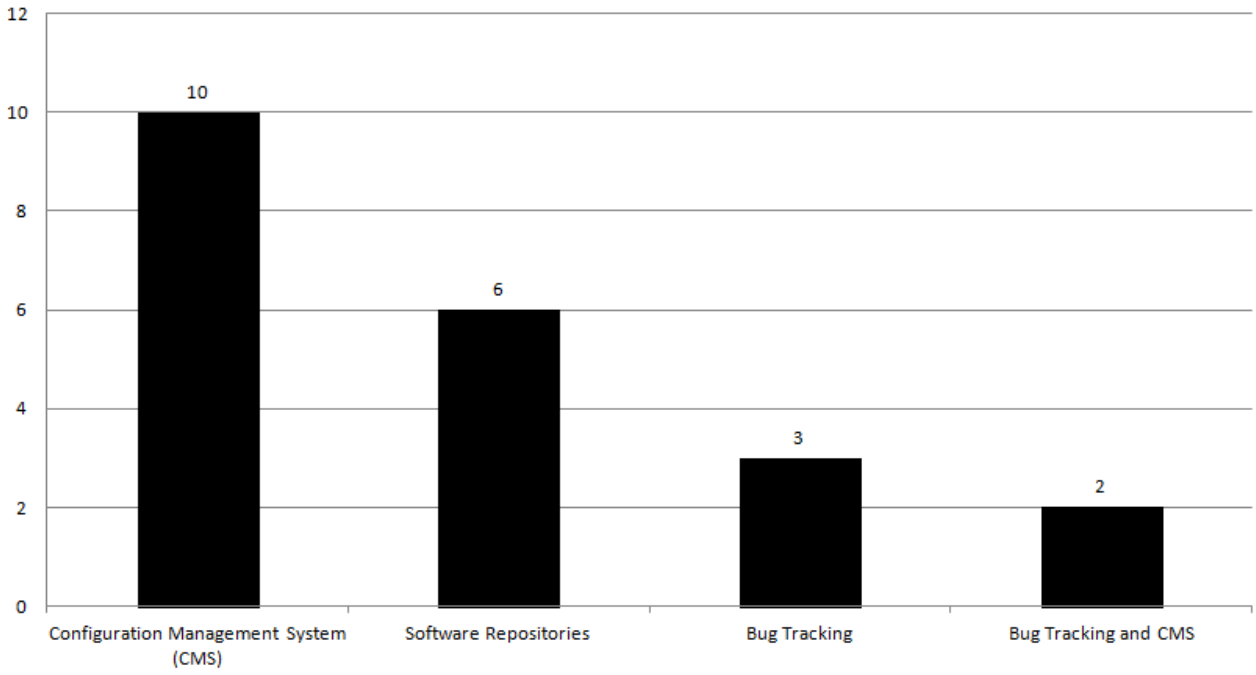

Figure 9. Papers by identified data source

5.2.3 Software Visualization Techniques

Only 6 of the 100 primary studies proposed software visualization techniques in the context of TD identification, indicating a fairly low use of visualization in this area. The types of software visualization techniques proposed to identify TD were: Flags in code, 2D maps, scatterplot and correlation matrix, time range, timeline, and treemap. Each type was cited only once.

Besides, the most common type of platform proposed to display visualizations is an automatic tool (with 4 citations). Spreadsheet was considered only on two papers. This is an expected result because TD identification activities usually collect a lot of data from the project, often more than can be easily handled in a spreadsheet, which is the only other platform used in studies involving visualization.

5.3. Strategies for Managing Technical Debt (Q3, Q3.1, Q3.2)

In this section, the TD management strategies found in the literature are presented, along with their definitions.

The only strategies with two or more references were:

- **Cost-Benefit Analysis**: This evaluates whether the expected interest is high enough to justify the payment of the debt. The interest rate is composed of two parts: the probability of interest and its value. The first part refers to the probability that the debt, if not paid, will result in extra cost to the project. The second part is an estimated amount of additional work that will be required if this item is not paid [Seaman et al., 2012];
- **Portfolio Approach**: The central concept of this strategy is the list of TD items. This list contains debt items identified for the project. The registration information is reported in a table that contains the location of the debt, the time at which it is identified, the responsible person, the reason why it is considered TD, an estimate of the principal, estimates of the expected interest amount (EIA) and interest standard deviation (ISD), and estimates of the correlations of this item with other TD items. During the planning of each software increment, an analysis of what should be paid and what can be postponed is done [Guo and Seaman, 2011];

- **Options**: For this strategy, investment in paying off the debt is analogous to purchasing the option that facilitates change to the software in the future, if the software has to be changed, but without immediate profits [Seaman et al., 2012];
- **Analytic Hierarchy Process (AHP):** AHP provides a method for structuring a problem, comparing alternatives with regard to specified criteria, and determining an overall ranking for each alternative. When applied to TD, the decision alternatives would be the various identified instances of technical debt, and the outcome of the strategy would be a prioritized ranking of these items, indicating which should be paid off first [Seaman et al., 2012];
- **Calculation of TD Principal**: The strategy focuses on the estimated principal. The objective is to use a defined process to estimate the TD Principal and to associate the identified issues with different quality attributes (ISO 9126). According to the authors, having the principal associated with quality attributes drives managers to make better decisions [Curtis, 2012];
- **Marking of Dependencies and Code Issues**: This is a strategy used to manage problems and dependencies in the project source code. The objective is to insert tags in the project's source in a way that is easy for the development team to visualize where TD is inserted and thus decide when to pay it, based on the involved effort and the availability of project time [Holvitie and Leppanen, 2013].

For more information on other management techniques, note the references in Table 5. In the table we can see that the management strategies "**Portfolio Approach"** and "**Cost Benefit Analysis"** were the most cited. But as the number of papers is still small, and mostly the strategies have not been evaluated, it is not possible to say whether the two strategies are more effective than other strategies.

Table 5. References of each management strategy

Management Strategy (Approaches, Methods and Models)	References	Number of Papers
Cost-Benefit Analysis	[Guo et al., 2011] [Seaman et al., 2012] [Stochel, 2012] [Holvitie and Ville, 2013] [Monteith and McGregor, 2013] [Griffith et al., 2014] [Alzaghoul and Bahsoon, 2014] [Ojameruaye and Bahsoon, 2014] [Ramasubbu and Kemerer, 2014] [Guo et al., 2014] [Holvitie, 2014]	11
Portfolio Approach	[Guo and Seaman, 2011] [Seaman et al., 2012] [Stochel, 2012] [Snipes et al., 2012] [Power, 2013] [Zazworka et al., 2013] [Ken, 2013] [Griffith et al., 2014] [Alzaghoul and Bahsoon, 2014] [Ojameruaye and Bahsoon, 2014]	10
Options	[Zazworka et al., 2011] [Seaman et al., 2012] [Alzaghoul and Bahsoon, 2013] [Griffith et al., 2014] [Alzaghoul and Bahsoon, 2014] [Ojameruaye and Bahsoon, 2014]	6

Analytic Hierarchy Process	[Seaman et al., 2012] [Alzaghoul and Bahsoon, 2014] [Ojameruaye and Bahsoon, 2014]	3
Calculation of TD-Principal	[Curtis et al., 2012] [Curtis et al., 2012b]	2
Marking of dependencies and Code Issues	[Morgenthaler et al., 2012] [Tom et al., 2013]	2
Debt Symptoms Index	[Marinescu, 2012]	1
Empirical Model of Technical Debt and Interest	[Nugroho et al., 2011]	1
Formal Approach to Technical Debt Decision Making	[Schmid, 2013]	1
Game Theoretic Competitive Source Control Approach	[Morrison-Smith et al., 2012]	1
Measuring symptom severity on a smell thermometer	[Ligu et al., 2013]	1
Metric for Managing Architectural Technical Debt	[Nord et al., 2012]	1
RE-KOMBINE Model	[Ernst, 2012]	1
SQALE Method	[Letouzey and Ilkiewicz, 2012]	1
Supply Chain Management	[Alzaghoul and Bahsoon, 2013]	1
Framework for Estimating Interest	[Singh et al., 2014]	1
Managing TD in database schemas	[Weber et al., 2014]	1
Benchmarking-Based Model	[Mayr et al., 2014]	1
Model for optimizing technical debt	[Ramasubbu and Kemerer, 2013]	1
Finance and accounting practices	[Conroy, 2012]	1

The number of published papers per year discussing management strategies was also investigated. In 2010, no management strategy was identified in any published paper. Only a handful of TD management papers were published in 2011, introducing such ideas as cost-benefit analysis and portfolio management. In 2012, the number of papers, and the number of newly introduced management techniques, exploded, with 16 papers published that year. In 2013, a total of 10 TD management papers were identified. Finally, in 2014, the number of papers exploring TD management was also high reaching 17 papers.

5.3.1 Evaluation Studies

Figure 10 shows the number of papers on management strategies and the type of performed evaluation. It can be observed that the studies are concentrated on case study method. Besides, not all strategies were evaluated. This implies that most of the proposals in the TD management area still require more experiments, so that their benefits and limitations can be known with increased confidence.

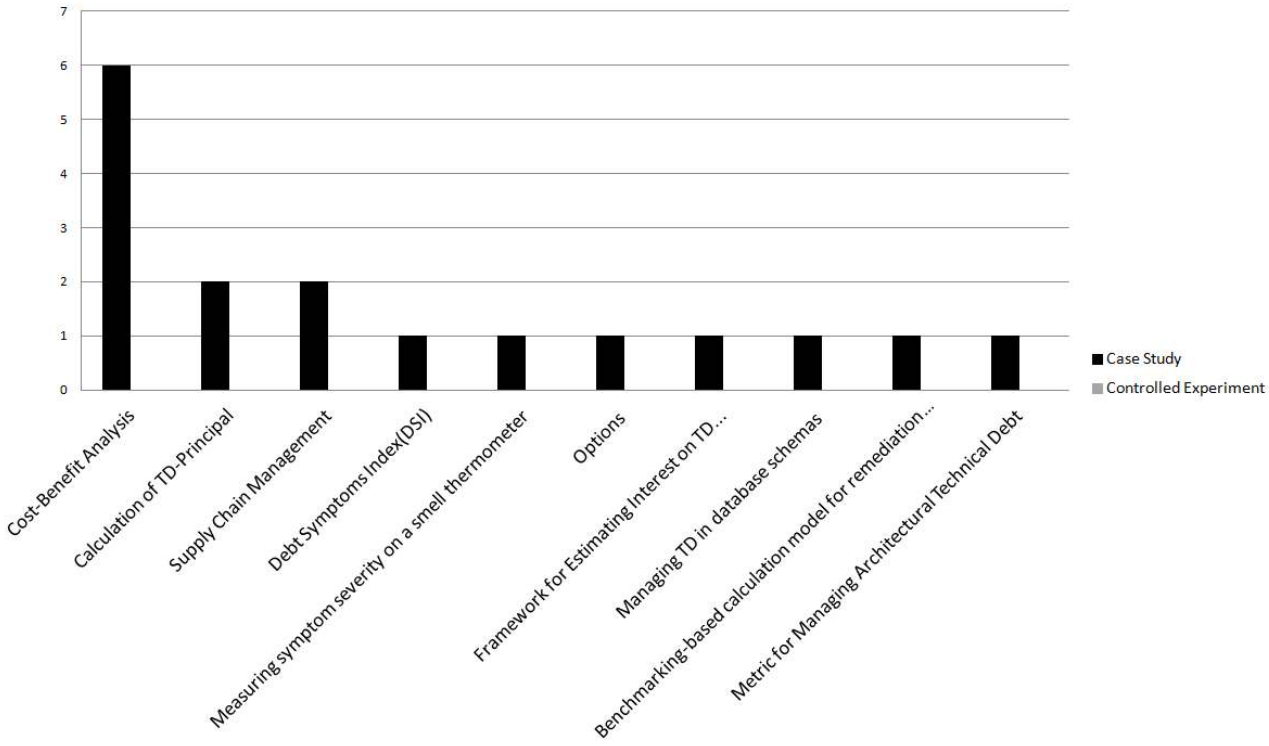

Figure 10. Papers by management strategy and type of study.

5.3.2 Software visualization techniques

As can be seen in Figure 11, 22 of the 100 primary studies proposed software visualization techniques in the context of TD management. The most proposed visualization techniques were dependency matrix, bar graph, and pie chart format. One opportunity that arises here is to investigate different types of software visualization techniques already proposed in other contexts in software maintenance and evolution [Novais et al. 2013], and adapt them to manage TD.

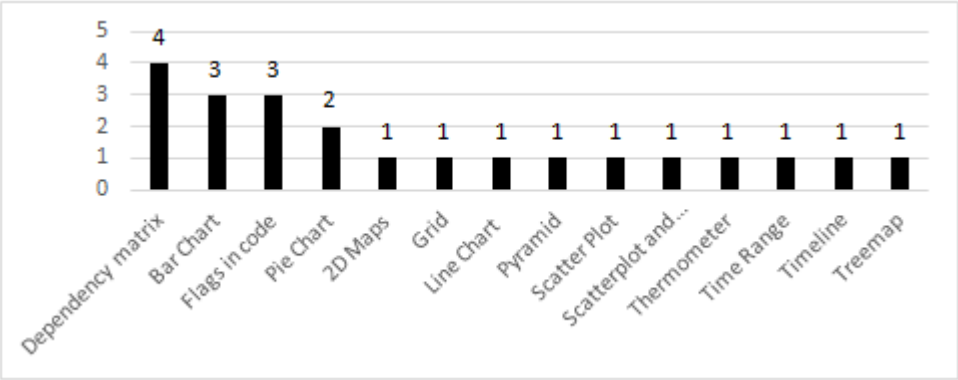

Figure 11. Types of software visualization techniques proposed to manage TD.

In complement, the most common type of platform proposed to display visualizations is the spreadsheet (with 15 citations). Automatic tool had only 7 citations. This is interesting in contrast to results of software visualization for TD identification, which shows that automatic tools are more prevalent and that papers describing visualization for TD identification are many fewer than those for TD management. This kind of manual solution, a spreadsheet, is far from ideal, as it requires a lot of effort to record the data extracted from the software project and keep it up-to-date.

6. Discussion

In this section, we compare our results to those of other secondary studies in this area, discuss and synthesize the results presented in section 5, and present the implications of these results for both researchers and practitioners.

6.1. Comparison to related work

In this section, we will discuss how this work differs from the related work introduced in Section 2. Table 6 shows the correspondence among the research questions found in each study.

Regarding "*(Q1) What are the types of TD?*", there is a partial overlapping among the studies. The overlapping is associated with the goal of the question, but looking into the results from each study, it is possible to observe a complementary relationship between them. Tom et al. (2012) classified debt into seven elements and, after, Tom et al. (2013) categorized them into five dimensions. The work of [Li et al., 2015] and our mapping study have 10 types in common but differ on 5 that were only identified in this work. Figure 12 represents the intersection among the studies. Thus, our proposed taxonomy complements other relevant categorizations of TD performed in the last years [Tom et al., 2013][Li et al., 2015]. However, the identified types of debt still require some sort of evaluation by the research community and software practitioners.

Table 6. Correspondence among research questions.

	Our Mapping Study	[Tom et al., 2012]	[Villar and Matalonga, 2013]	[Tom et al., 2013]	[Ampatzoglou et al., 2015]	[Li et al., 2015]
Research Questions	**(Q1)** What are the types of TD?	**(RQ1)** What are the elements of technical debt?	-	**(RQ1)** What are the dimensions of technical debt?	-	**(RQ1)** What are the types of TD and what is not considered as TD? **(RQ2)** What TD types are researchers and practitioners mostly working on and what types are under-studied?
	(Q2) What are the strategies proposed to identify TD?	-	-	-	-	-
	(Q2.1) Which empirical evaluations have been performed?	-	-	-	-	-
	(Q2.2) Which artifacts and data sources have been proposed to identify the TD?	-	-	-	-	-
	(Q2.3) Which software visualization techniques have been proposed to identify TD?	-	-	-	-	-
	(Q3) What strategies have been proposed for the management of TD?	-	-	-	**(RQ2)** Which financial approaches have been applied in technical debt management?	**(RQ6)** What are the different activities of TDM? **(RQ7)** What approaches are used in each TDM activity? **(RQ8)** What tools are used in TDM and what TDM activities are supported by these tools?
	(Q3.1) Which empirical evaluations have been performed?					

		(Q3.2) Which software visualization techniques have been proposed to manage TD?	-	-	-	**-**	**-**
Papers Published Until	2014		2012 (partially)	2012 (partially)	2012 (partially)	2013 (partially)	2013

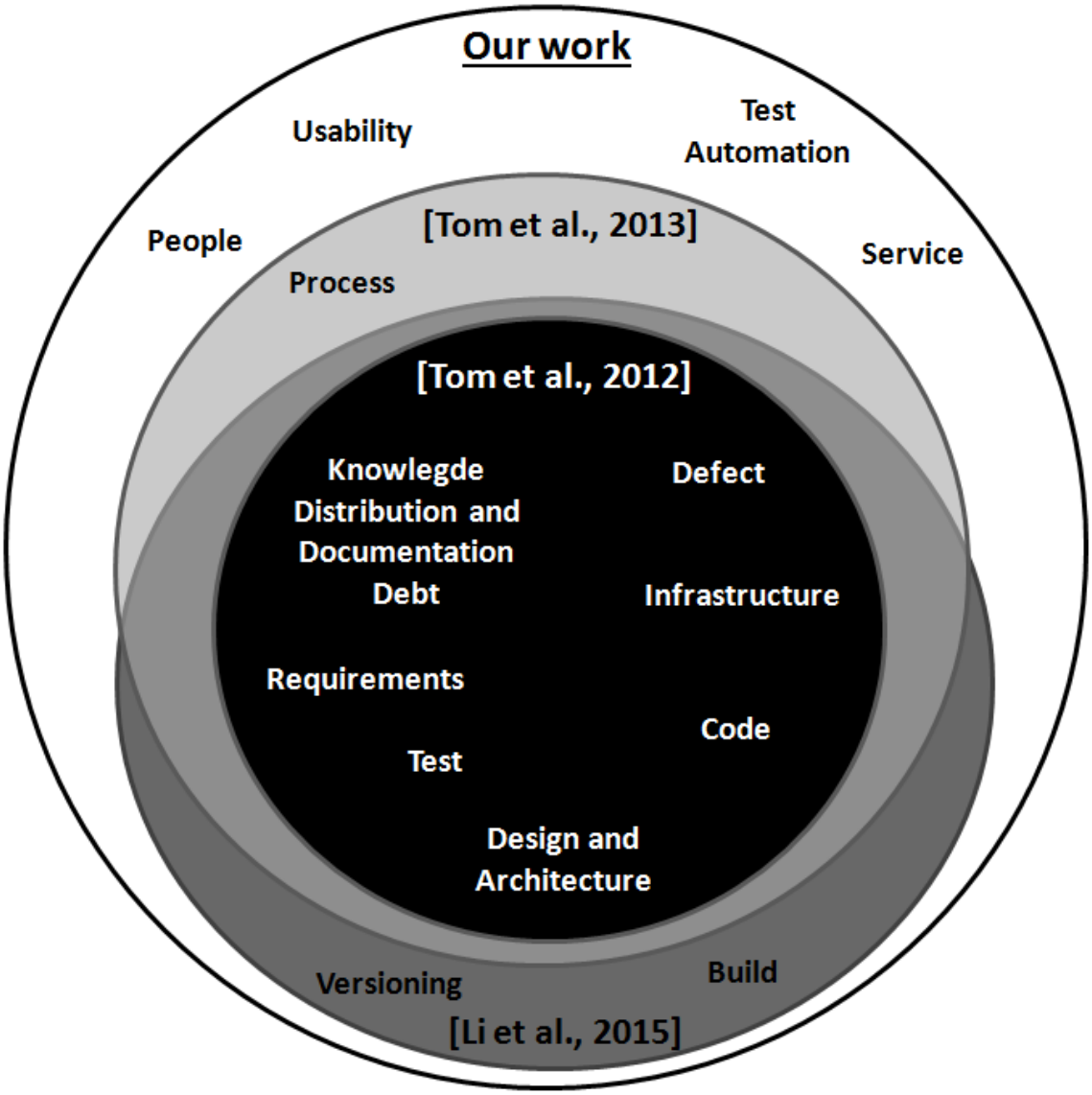

Figure 12. Intersection among the studies for Research Question 1 (types of TD)

The goal of our second research question, "*(Q2) What are the strategies proposed to identify TD?*" and its sub-questions (*Q2.1*, *Q2.2*, and *Q2.3*) is not addressed by any of the other secondary studies.

For "*(Q3) What strategies have been proposed for the management of TD?*", both the goals and results of each study are different. Ampatzoglou et al. (2015) presented a list of finance-based TD management strategies. In comparison to our work, we reached a more comprehensive list of management strategies (including the strategies found by Ampatzoglou et al. (2015)) because we considered any TD management strategy. The work of Li et al. (2015) identified activities that are performed (for instance: TD identification, repayment, prevention, communication, and monitoring) to manage TD. Our work focuses on strategies that have been proposed to support the TD management (for instance: cost-benefit analysis, portfolio approach, and SQALE Method). Thus, this research question has a strong complementary relationship among the studies and weak overlapping between them as can be seen in Table 7. Besides, our research question "*(Q3.2) Which software visualization techniques have been proposed to manage TD?*" does not overlap with any of the research performed in

the other related work.

Table 7. Intersection among the studies for Research Question 3 (TD Management)

		Our Mapping Study	[Ampatzoglou et al., 2015]	[Li et al., 2015]
TD Management Approaches	Cost-Benefit Analysis	x	x	
	Portfolio Approach	x	x	
	Options	x	x	
	Analytic Hierarchy Process	x	x	
	Calculation of TD-Principal	x		
	Marking of dependencies and Code Issues	x		
	Debt Symptoms Index	x		
	Empirical Model of Technical Debt and Interest	x	x	
	Formal Approach to Technical Debt Decision Making	x		
	Game Theoretic Competitive Source Control Approach	x		
	Measuring symptom severity on a smell thermometer	x		
	Metric for Managing Architectural Technical Debt	x		
	RE-KOMBINE Model	x		
	SQALE Method	x		
	Supply Chain Management	x		
	Framework for Estimating Interest	x		
	Managing TD in database schemas	x		
	Benchmarking-Based Model	x		
	Model for optimizing technical debt	x	x	
	Finance and accounting practices	x	x	
TD Management	Prevention			x
	Communication			x
	Representation/Documentation			x

	Repayment		**x**	**x**
	Monitoring		**x**	**x**
	Prioritization		**x**	**X**
	Measurement		**x**	**X**
	Identification		**x**	**x**

Regarding "*(Q2.1 and Q3.1) Which empirical evaluations have been performed?*", there are no similar research questions in the other studies. However, we analyzed each of the other studies more closely in order to identify possible intersections. Li et al. (2015) was the only study that took into consideration if the identified technique had been evaluated. The authors used this information to represent a general view of the evidence levels of their selected studies. In contrast, in our work we used this information to characterize the level of confidence we have about TD indicators and TD management strategies.

Finally, the mapping study presented in this work considers papers published up to 2014. Specifically, this means 27 additional papers published in the last year selected for analysis.

6.2. Synthesis of the results

In this section, we present a discussion of the results. The objective of this systematic mapping was to identify strategies that have been proposed to identify or manage TD in software projects. For this, we analyzed 100 studies in order to identify types of TD, strategies proposed to identify them, and strategies proposed to support their management.

In order to consolidate the results achieved in the mapping, Figure 13 details the relationship between the number of analyzed papers by subject (indicators, artifacts, data sources, management strategy, and software visualization), the types of TD and the number of performed empirical studies. Through this figure, one can observe that most studies deal with TD at the source code level, i.e. design, defect, code and architecture debt (i.e. in the upper left area of the bubble chart).

It is also possible to observe that there is debt throughout the whole lifecycle of the project. Thus, ensuring the quality of the project's source code is not the only way to enhance project quality. However, despite many types of TD, much of the work focuses on studying existing problems in the source code. The emphasis on source code in TD research may be explained by the fact that there are already a set of metrics and automated support tools to extract information that can be used as indicators of TD (e.g. ASA Issues, Code Smells, among others). Another possible explanation is that the body of knowledge in TD is still consolidating itself. For example, the TD types infrastructure, process, service, and test automation debt were only first cited in 2013. These newer types are not yet fully vetted by the community. Thus, it is too soon to say whether these newer types will be accepted as "real" types of TD, or a case of stretching the metaphor too far. If they appear to be useful categorizations of debt, then work will proceed on developing indicators and tools and techniques to use those indicators. Management of these new types of TD will then be addressed.

We were able to identify several studies on strategies to manage TD. However,

although this is a considerable number, only five strategies (Portfolio Approach, Cost-Benefit Analysis, Analytic Hierarchy Process, Calculation of TD Principal, and Marking of dependencies and Code Issues) were cited in more than two papers and few of them were evaluated. This shows that most of the authors propose new strategies, but few are conducting studies to evaluate their real applicability.

We can also see in Figure 13 that there are still few studies conducted on using software visualization techniques to support TD related activities. However, we believe that the use of software visualization techniques can facilitate the work of identification and management of TD in the evolution of software projects.

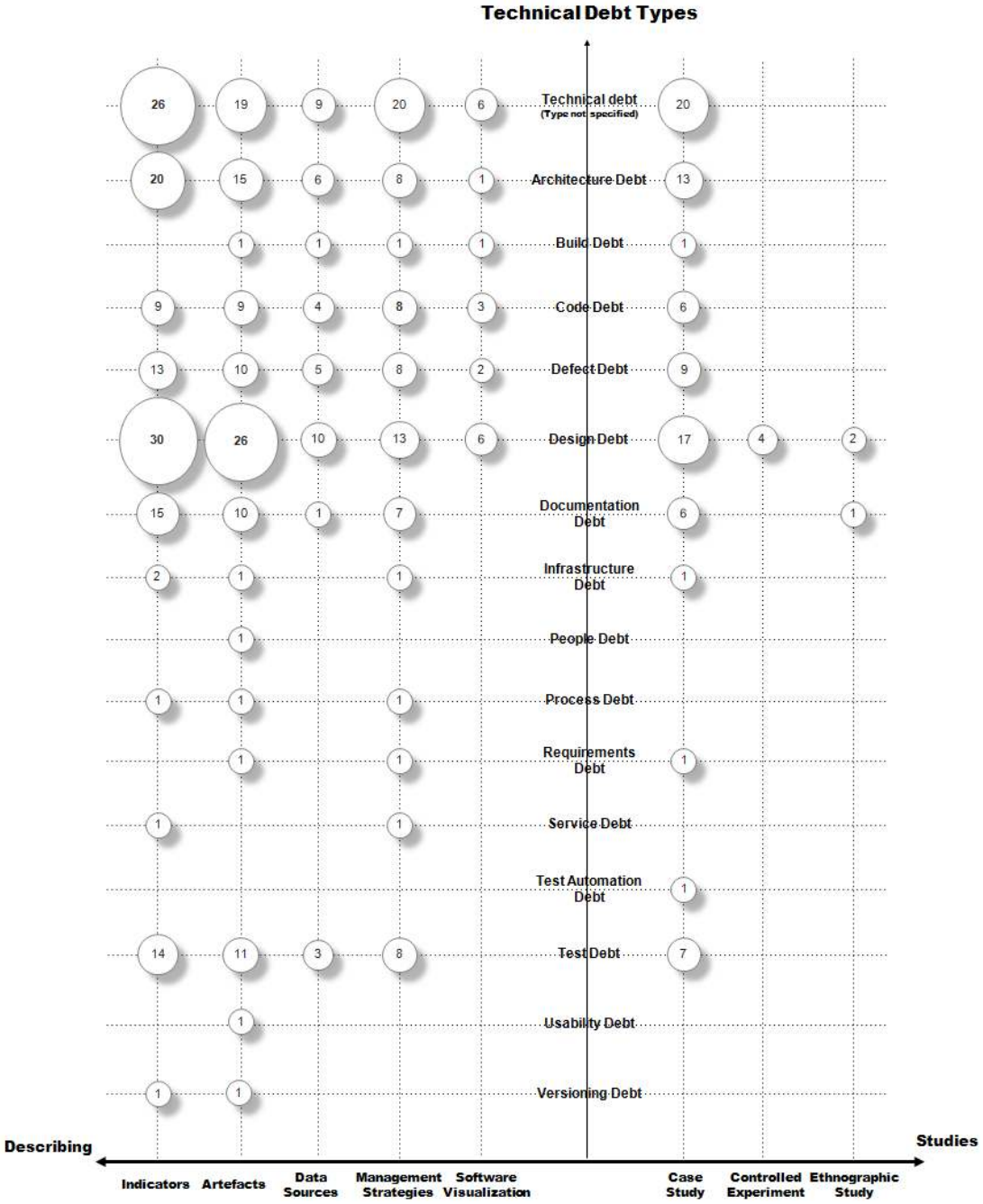

Figure 13. A visualization of the systematic mapping in the form of a bubble chart

Furthermore, when looking at the right side of the graph in Figure 13, where the empirical studies in the area are presented, we can see that our knowledge regarding the

real benefits and limitations of what has been proposed by the research community is still limited. Figure 14 summarizes the number of studies by research topic. The categories used to group the research topics were extracted from the goal of each analyzed empirical study:

- **Evolution**: research that investigates the behavior of the TD items during the evolution of the software;
- **Identification**: studies whose focus is on the definition of processes/activities and indicators that allow the discovery of TD items in software projects;
- **Management**: papers that propose strategies to measure the quantity and value of incurred TD, as well as criteria to define the best moment to pay the debt;
- **Perception of TD**: studies that investigate TD in a more generic perspective, dealing with issues such as the developer's perception of TD.

Our results show that there is a balance between efforts aimed at the evaluation of TD management and TD identification strategies. This result is expected, since these activities are the first ones that the development team has to consider when servicing the debt in its projects. However, little effort has been spent on studies related to the evolution of debt during the development and maintenance of the project.

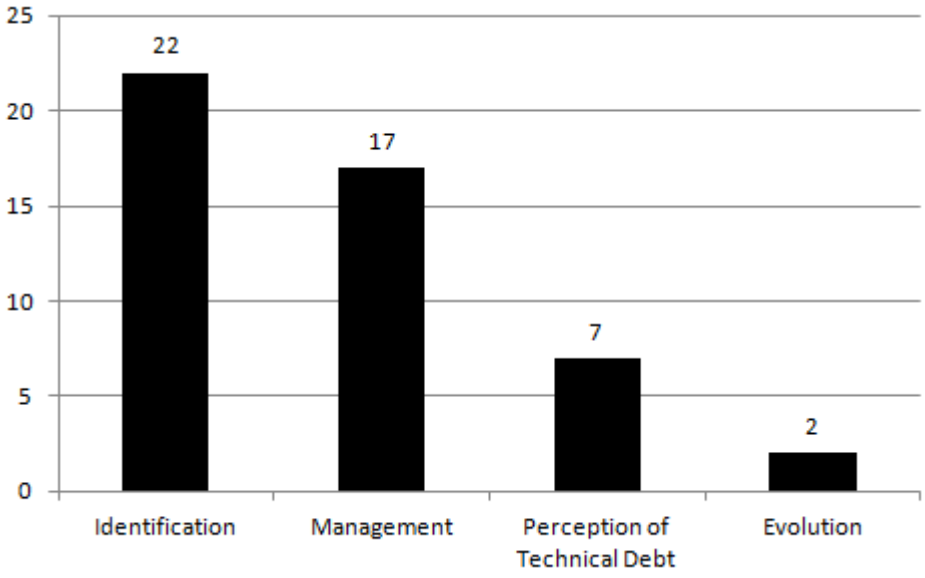

Figure 14. Papers by research topic.

6.3 Implications for Practitioners and Researchers

This mapping study has primarily focused on facilitating and guiding future research in TD identification and management. However, our results have important implications for practitioners as well, particularly those looking to the literature for guidance on how to manage TD on real projects:

- TD can be found in many different artifacts produced during the software development process. As consequence, a variety of strategies must be employed if the goal is to find all types of TD that might have a negative impact;
- There are several indicators for each type of TD. So developers have choices when defining a strategy to identify and track TD in their projects, and should define criteria for choosing indicators;

- Visualizing TD through software visualization techniques is still a hard task due to the lack of support tools and research;
- Most research regarding TD identification is code-related. This could suggest that focus on TD identification activities considering, initially, code-related debt (code, design, and architecture debt) would make sense. However, this must be assumed to be a risk because debt can be hidden in the project in different ways (not only code-related) and non-code-related debt can bring significant negative impacts to the project. Thus, our suggestion is to avoid the temptation to limit the focus to code-related debt;
- We also have identified several TD management strategies. Most of them still require further investigation and empirical evaluation. However, they can be a good starting point for customizing or defining a TD management strategy for a real software project.

For researchers, the findings of this mapping study point to the following implications:

- There are different types of TD and some indicators for each of them, but we have not identified any evidence on how to use this set of information to guide initiatives of TD identification in real settings. Despite progress in different areas of TD, there is still a need to take a look at the big picture and investigate holistic strategies to managing TD effectively in the software industry;
- Despite the fact that software visualization has been shown to benefit the process of software understanding, there is still little investigation relating TD and software visualization;
- Few empirical studies have been performed in real settings. This is an indicator that, for some areas, we still do not fully understand all the costs or benefits of the proposed TD indicators and management strategies. Many of these proposals require deeper investigation. Some of them were just cited in some papers;
- Research on TD is highly concentrated on a few types of debt (design, architecture, code, and defect) while other types are currently under-investigated. This shows a clear gap that could be explored in the coming years.

Our results clearly show an active and fruitful area of investigation that is continuing to grow and is still in need of maturation, in terms of consolidating concepts and empirically validating new proposals. In order for this body of work to present a useful contribution to practice, the research community must find ways to guide practitioners to those strategies most likely to be helpful in a particular context, and to adapt those strategies to a given situation.

7. Threats to validity

The results of this systematic mapping may have been affected by some threats to validity, such as:

- **Research Question**: The defined research questions in this study may not cover the entire area of TD. To address this risk, the defined questions were analyzed by at least two researchers, one of whom acted as an external reviewer of the protocol. In addition, the protocol was presented at the 1st Latin American School of Software Engineering[6] and evaluated by at least two other independent researchers. All comments were considered in defining the final list

[6] www.inf.ufrgs.br/elaes2013

of research questions.

- **Publication Bias**: It is difficult to ensure that all relevant work was returned as results in the performed searches. To minimize this threat, the main digital libraries in computing were considered.
- **Search string**: There are two main concerns regarding the search string. Both are related to the using of the term "technical debt" as part of the search string:
    - First, there are potentially some relevant studies published before the term "technical debt" was widely used. For example, god classes (a type of code smell) are considered a good indicator of design debt, but they existed before they were associated with design debt. Thus, there are some papers discussing god classes that do not mention TD in their text (and so are not included in our mapping study). However, the scope of this mapping study was limited to how those subjects have been discussed from the point of view of TD. Thus, rather than presenting a comprehensive view of what the technical literature has said about god classes, we were interested in how the research community relates god classes and TD;
    - Second: there is a risk of excluding some papers that just use the term "debt" or a particular type of debt without using the term "technical debt". To investigate the extent of this risk, before performing the search in this study, we tested the search string using the string ("debt" AND "software"). The result was too generic and returned a substantial number of papers (for example, 1738 and 5032 papers in digital libraries ACM Digital Library and Science Direct respectively) that were not related to the research goal. On the other side, for the search using the terms ("technical debt" AND "software"), the results were more restrictive (152 in ACM Digital Library and 34 in Science Direct). We took both datasets and did a manual search to evaluate if the more restrictive string was causing the loss of any relevant study. The result was negative. Despite the fact that we do not have any guarantee that this manual search is 100% accurate, we believe that it indicated that we could use the more restrictive search string.
- **Data Extraction**: it is difficult to ensure that all the relevant primary studies were selected for this mapping or that the returned studies were appropriately analyzed. To reduce this risk, the classification and extraction of information was performed by at least two researchers.

Finally, it is also important to consider that the term TD is new. Papers began to be published recently. On the other side, related research may have been performed before. As this is a complex variable to be assessed and cannot be easily controlled, in this study we chose to only consider studies if they were developed by the research community from the TD perspective.

8. Concluding remarks and future work

The goal of this work was to conduct a systematic mapping of the literature to investigate strategies that have been proposed to identify and manage TD in the software lifecycle. Therefore, through the mapping it was possible to provide a comprehensive view about the current state of TD research.

We have identified 100 primary studies, covering strategies, techniques and/or tools for dealing with TD in software projects. These strategies vary in terminology, descriptions, artifacts, indicators, management strategies, empirical studies, data sources, and software visualization techniques to identify and/or manage the TD. We summarized the results and created a spreadsheet for organizing and analyzing the collected data. At the end, we provide the following contributions:

(i) an improved version of the taxonomy published previously in [Alves et al., 2014];
(ii) a list of indicators used to support the identification of TD;
(iii) a list of proposed TD management strategies;
(iv) an analysis of the types of empirical evaluation performed on the studies;
(v) a list of data sources used in TD identification activities;
(vi) a list of software visualization techniques used to identify and manage TD.

Moreover, we have characterized the current state of the art in TD by identifying possible gaps and topics where new research efforts can be invested.

The study shows growing interest of researchers in the TD area. Further, the number of new proposals for TD indicators and types is also growing. However, empirical evaluation of these new proposals is lagging behind. This indicates that the TD research area is in a phase of expansion and innovation, but just beginning a phase of careful evaluation and narrowing down the field to the most effective practices. Further, new proposals for types of TD raise the need for new proposals for indicators to help find those types, and management strategies to control them. Thus, despite a good number of proposed strategies, it is necessary to conduct further studies in the area to investigate new techniques and tools that could support developers with the control of TD. In addition, is necessary to carry out more empirical studies to validate the strategies that have been proposed.

In our future research agenda, we will work with the research gaps identified in this paper. We intend to evaluate the proposed taxonomy by the research community and software practitioners. We also intend to combine the evidence identified in this work with new theories and empirical studies developed by our research group in order to design new methods and tools to support TD identification and management activities.

Acknowledgments

The authors would like to thank CAPES for the financial support to this work. This work was also partially supported by CNPq Universal 2014 grant 458261/2014-9. Dr. Shull's contribution to this material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0002227.

Appendix A. Papers identified in the systematic mapping study

Al Mamun, M.; Berger, C. & Hansson, J. (2014), Explicating, Understanding, and Managing Technical Debt from Self-Driving Miniature Car Projects, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 11-18.

Allman, E. (2012), Managing Technical Debt, Queue, 10(3).

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

Alzaghoul, E. & Bahsoon, R. (2013), Economics-Driven Approach for Managing Technical Debt in Cloud-Based Architectures, in Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on, pp. 239-242.

Alzaghoul, E. & Bahsoon, R. (2014), Evaluating Technical Debt in Cloud-Based Architectures Using Real Options, in Software Engineering Conference (ASWEC), 2014 23rd Australian, pp. 1-10.

Barton, B. & Sterling, C. (2010), Manage Project Portfolios More Effectively by Including Software Debt in the Decision Process, Cutter IT Journal, Vol. 23, No. 10, 19-24.

Bohnet, J. & Dцllner, J. (2011), Monitoring code quality and development activity by software maps, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Brondum, J. & Zhu, L. (2012), Visualising architectural dependencies, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 7-14.

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. & Zazworka, N. (2010), Managing technical debt in software-reliant systems, FoSER 10: Proceedings of the FSE/SDP workshop on Future of software engineering research.

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

Conroy, P. (2012), ‘‘Technical Debt: Where Are the Shareholders’ Interests?’’, Software, IEEE Computer Society, 29 (6), p. 88, November/December 2012.

Curtis, B.; Sappidi, J. & Szynkarski, A. (2012), Estimating the Principal of an Applications Technical Debt, Software, IEEE 29(6), 34-42.

Curtis, B.; Sappidi, J. & Szynkarski, A. (2012), Estimating the size, cost, and types of Technical Debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 49-53.

Davis, N., (2013), Driving Quality Improvement and Reducing Technical Debt with the Definition of Done, Syst., Pittsburgh, PA, USA, Agile Conference (AGILE), pp. 164-168.

de Groot, J.; Nugroho, A.; Back, T. & Visser, J. (2012), What is the value of your software?, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 37-44.

Ernst, N. (2012), On the role of requirements in understanding and managing technical debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 61-64.

Falessi, D.; Shaw, M.; Shull, F.; Mullen, K. & Keymind, M. (2013), Practical considerations, challenges, and requirements of tool-support for managing technical debt, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp.

16-19.

Fontana, F.; Ferme, V. & Spinelli, S. (2012), Investigating the impact of code smells debt on quality code evaluation, in, pp. 15-22.

Gat, I. & Heintz, J. D. (2011), From assessment to reduction: how cutter consortium helps rein in millions of dollars in technical debt, MTD'11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Giraldo, F.; España, S.; Pineda, M.; Giraldo, W. & Pastor, O. (2014), Integrating technical debt into MDE, CEUR Workshop Proceedings 1164, 145-152.

Gomes, R.; Siebra, C.; Tonin, G.; Cavalcanti, A.; Silva, F. Q. D.; Santos, A. L. & Marques, R. (2011), An extraction method to collect data on defects and effort evolution in a constantly modified system, MTD'11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Greening, Daniel R. (2013), Release Duration and Enterprise Agility, 46th Hawaii International Conference on System Sciences (HICSS), pp. 4835-4841.

Griffith, I.; Izurieta, C.; Taffahi, H. & Claudio, D. (2014), A Simulation Study of Practical Methods for Technical Debt Management in Agile Software Development, in Proceedings of the 2014 Winter Simulation Conference, IEEE Press, Piscataway, NJ, USA, pp. 1014--1025.

Griffith, I.; Reimanis, D.; Izurieta, C.; Codabux, Z.; Deo, A. & Williams, B. (2014), The Correspondence Between Software Quality Models and Technical Debt Estimation Approaches, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 19-26.

Guo, Y. & Seaman, C. (2011), A portfolio approach to technical debt management, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Guo, Y.; Seaman, C.; Gomes, R.; Cavalcanti, A.; Tonin, G.; da Silva, F.; Santos, A. L. M. & Siebra, C. (2011), Tracking technical debt - An exploratory case study, in Software Maintenance (ICSM), 2011 27th IEEE International Conference on, pp. 528-531.

Guo, Y.; Seaman, C.; Zazworka, N. & Shull, F. (2010), Domain-specific tailoring of code smells: an empirical study, ICSE 10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering.

Guo, Y.; Spínola, R. & Seaman, C. (2014), Exploring the costs of technical debt management – a case study, Empirical Software Engineering, 1-24.

Ho, J. & Ruhe, G. (2014), When-to-Release Decisions in Consideration of Technical Debt, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 31-34.

Holvitie, J. & Leppanen, V. (2013), DebtFlag: Technical debt management with a development environment integrated tool, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 20-27.

Holvitie, J. (2014), Software implementation knowledge management with technical debt and network analysis, in Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on, pp. 1-6.

Holvitie, J.; Laakso, M.-J.; Rajala, T.; Kaila, E. & Leppänen, V. (2013), The Role of Dependency Propagation in the Accumulation of Technical Debt for Software

Implementations, in Ákoss Kiss, ed., 13th Symposium on Programming Languages and Software Tools, University of Szeged, pp. 61–75.

Holvitie, J.; Leppanen, V. & Hyrynsalmi, S. (2014), Technical Debt and the Effect of Agile Software Development Practices on It - An Industry Practitioner Survey, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 35-42.

Izurieta, C.; Griffith, I.; Reimanis, D. & Luhr, R. (2013), On the Uncertainty of Technical Debt Measurements, in Information Science and Applications (ICISA), 2013 International Conference on, pp. 1-4.

Izurieta, C.; Vetro, A.; Zazworka, N.; Cai, Y.; Seaman, C. & Shull, F. (2012), Organizing the technical debt landscape, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 23-26.

Kaiser, M. & Royse, G. (2011), Selling the Investment to Pay Down Technical Debt: The Code Christmas Tree, in Agile Conference (AGILE), 2011, pp. 175-180.

Klinger, T.; Tarr, P.; Wagstrom, P. & Williams, C. (2011), An enterprise perspective on technical debt, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Krishna, V. & Basu, A. (2012), Minimizing Technical Debt: Developers viewpoint, in Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012), International Conference on, pp. 1-5.

Krishna, Vinay; Basu, A. (2013), Software Engineering Practices for Minimizing Technical Debt, Proceedings of the International Conference on Software Engineering Research and Practice (SERP), 1-5.

Kruchten, P.; Nord, R. & Ozkaya, I. (2012), Technical Debt: From Metaphor to Theory and Practice, Software, IEEE 29(6), 18-21.

Ktata, O. & Lévesque, G. (2010), Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?, C3S2E 10: Proceedings of the Third C* Conference on Computer Science and Software Engineering.

Letouzey, J. & Ilkiewicz, M. (2012), Managing Technical Debt with the SQALE Method, Software, IEEE 29(6), 44-51.

Letouzey, J.-L. (2012), The SQALE method for evaluating Technical Debt, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 31-36.

Li, Z.; Liang, P.; Avgeriou, P.; Guelfi, N. & Ampatzoglou, A. (2014), An Empirical Investigation of Modularity Metrics for Indicating Architectural Technical Debt, in Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures, ACM, New York, NY, USA, pp. 119-128.

Ligu, E.; Chatzigeorgiou, A; Chaikalis, T.; Ygeionomakis, N. (2013), Identification of Refused Bequest Code Smells, Dept. of Appl. Inf., Univ. of Macedonia, Thessaloniki, Greece, 29th IEEE International Conference on Software Maintenance (ICSM), pp. 392-395.

Lim, E.; Taksande, N. & Seaman, C. (2012), A Balancing Act: What Software Practitioners Have to Say about Technical Debt, Software, IEEE 29(6), 22-27.

Lindgren, M. (2012), Bridging the software quality gap, Department of Computing Science, Umea University.

Marinescu, R. (2012), Assessing technical debt by identifying design flaws in software systems, IBM Journal of Research and Development 56(5), 9:1-9:13.

Martini, A.; Bosch, J. & Chaudron, M. (2014), Architecture Technical Debt: Understanding Causes and a Qualitative Model, in Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, pp. 85-92.

Mayr, A.; Plosch, R. & Korner, C. (2014), A Benchmarking-Based Model for Technical Debt Calculation, in Quality Software (QSIC), 2014 14th International Conference on, pp. 305-314.

McGregor, J. D.; Monteith, J. & Zhang, J. (2012), Technical debt aggregation in ecosystems, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 27-30.

Mo, R.; Garcia, J.; Cai, Y. & Medvidovic, N. (2013), Mapping architectural decay instances to dependency models, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 39-46.

Monteith, J. & McGregor, J. (2013), Exploring software supply chains from a technical debt perspective, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 32-38.

Morgenthaler, J.; Gridnev, M.; Sauciuc, R. & Bhansali, S. (2012), Searching for build debt: Experiences managing technical debt at Google, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 1-6.

Morrison-Smith, S.; Dighans, S.; Daniels, T.; Marmon, C. & Izurieta, C. (2012), Technical debt reduction using a game theoretic competitive source control approach, in , pp. 157 - 162.

Nacimento, C.; Matalonga, S. & Rossa Hauck, J. (2014), Identifying technical debt cost factors in reflection activities of an agile projects, in Computing Conference (CLEI), 2014 XL Latin American, pp. 1-11.

Neill, C. & Laplante, P. (2006), Paying down design debt with strategic refactoring, Computer 39(12), 131-134.

Nord, R.; Ozkaya, I.; Kruchten, P. & Gonzalez-Rojas, M. (2012), In Search of a Metric for Managing Architectural Technical Debt, in Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, pp. 91-100.

Nugroho, A.; Visser, J. & Kuipers, T. (2011), An empirical model of technical debt and interest, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

OConnor, D. (2010), Technical debt in semiconductor equipment: Its time to pay it down, Solid State Technology 53(7), 34-35.

Ojameruaye, B. & Bahsoon, R. (2014), Systematic Elaboration of Compliance Requirements Using Compliance Debt and Portfolio Theory, in Camille Salinesi & Inge van de Weerd, ed., Requirements Engineering: Foundation for Software Quality, Springer International Publishing, pp. 152-167.

Olbrich, S.K., Cruzes, D.S., and Sjoberg, D. I. K. (2010), "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in Proceedings of the 2010 IEEE International Conference on Software

Maintenance, ser. ICSM ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2010.5609564

Potdar, A. & Shihab, E. (2014), An Exploratory Study on Self-Admitted Technical Debt, in Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on, pp. 91-100.

Power, K. (2013), Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 28-31.

Prause, C. R. (2011), Reputation-based self-management of software process artifact quality in consortium research projects, ESEC/FSE’11: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.

Pugh, K. (2010), The Risks of Acceptance Test Debt, Cutter IT Journal, Vol. 23, No. 10, 23-29.

Ramasubbu, N. & Kemerer, C. (2013), Towards a model for optimizing technical debt in software products, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 51-54.

Ramasubbu, N. & Kemerer, C. (2014), Managing Technical Debt in Enterprise Software Packages, Software Engineering, IEEE Transactions on 40(8), 758-772.

Schmid, K. (2013), A formal approach to technical debt decision making, QoSA 13: Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures.

Schmid, K. (2013), On the limits of the technical debt metaphor some guidance on going beyond, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 63-66.

Schumacher, J.; Zazworka, N.; Shull, F.; Seaman, C. & Shaw, M. (2010), Building empirical support for automated code smell detection, ESEM’10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.

Seaman, C. & Guo, Y. (2011), Measuring and Monitoring Technical Debt, Advances in Computers 82, 25-46.

Seaman, C.; Guo, Y.; Zazworka, N.; Shull, F.; Izurieta, C.; Cai, Y. & Vetro, A. (2012), Using technical debt data in decision making: Potential decision approaches, in Managing Technical Debt (MTD), 2012 Third International Workshop on, pp. 45-48.

Shafer, A. C. (2010), Infrastructure Debt: Revisiting the Foundation, Cutter IT Journal, Vol. 23, No. 10, 36-40.

Shah, S.; Torchiano, M.; Vetro, A. & Morisio, M. (2013), Exploratory testing as a source of testing technical debt, IT Professional, pp. (99), 1-1.

Shah, S.; Torchiano, M.; Vetro, A. & Morisio, M. (2014), Exploratory Testing as a Source of Technical Debt, IT Professional 16(3), 44-51.

Sharma, T. (2012), Quantifying Quality of Software Design to Measure the Impact of Refactoring, in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, pp. 266-271.

Shull, F. (2011), Perfectionists in a World of Finite Resources, Software, IEEE 28(2), 4-6.

Siebra, C. A.; Tonin, G. S.; Silva, F. Q.; Oliveira, R. G.; Junior, A. L.; Miranda, R. C. & Santos, A. L. (2012), Managing technical debt in practice: an industrial report, ESEM 12: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement.

Siebra, C.; Cavalcanti, A.; Silva, F.; Santos, A. & Gouveia, T. (2014), Applying Metrics to Identify and Monitor Technical Debt Items during Software Evolution, in Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pp. 92-95.

Singh, V.; Snipes, W. & Kraft, N. (2014), A Framework for Estimating Interest on Technical Debt by Monitoring Developer Activity Related to Code Comprehension, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 27-30.

Skourletopoulos, G.; Bahsoon, R.; Mavromoustakis, C.; Mastorakis, G. & Pallis, E. (2014), Predicting and quantifying the technical debt in cloud software engineering, in Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on, pp. 36-40.

Snipes, W.; Robinson, B.; Guo, Y. & Seaman, C. (2012), Defining the decision factors for managing defects: A technical debt perspective, in, pp. 54-60.

Spínola, R.; Zazworka, N.; Vetro`, A.; Seaman, C. & Shull, F. (2013), Investigating technical debt folklore: Shedding some light on technical debt opinion, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 1-7.

Stephen Chin, Erik Huddleston, W. B. & Gat, I. (2010), The Economics of Technical Debt, Cutter IT Journal, Vol. 23, No. 10, 11-15.

Stochel, M.; Wawrowski, M. & Waskiel, J. (2012), Adaptive Agile Performance Modeling and Testing, in Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual, pp. 446-451.

Theodoropoulos, T.; Hofberg, M. & Kern, D. (2011), Technical debt from the stakeholder perspective, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Tom, E.; Aurum, A. & Vidgen, R. b. (2013), An exploration of technical debt, Journal of Systems and Software 86(6), 1498-1516.

Tom, E.; Aurum, A. & Vidgen, R. T. (2012), A Consolidated Understanding of Technical debt, in 20th European Conference on Information Systems, ECIS 2012, Barcelona, Spain, June 10-13, 2012, pp. 16.

Vetro, A. (2012), Using automatic static analysis to identify technical debt, ICSE 2012: Proceedings of the 2012 International Conference on Software Engineering.

Wang, P.; Yang, J.; Tan, L.; Kroeger, R. & David Morgenthaler, J. (2013), Generating precise dependencies for large software, in Managing Technical Debt (MTD), 2013 4th International Workshop on, pp. 47-50.

Weber, J.; Cleve, A.; Meurice, L. & Bermudez Ruiz, F. (2014), Managing Technical Debt in Database Schemas of Critical Software, in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, pp. 43-46.

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

Xuan, J.; Hu, Y. & He, J. (2012), Debt-prone bugs: Technical debt in software maintenance, International Journal of Advancements in Computing Technology 4(19), 453-461.

Yli-Huumo, J.; Maglyas, A. & Smolander, K. (2014), The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company, in 15th International Conference, PROFES 2014, Springer International Publishing, pp. 93-107.

Zazworka, N. b.; Vetro, A. c.; Izurieta, C.; Wong, S.; Cai, Y.; Seaman, C. g. & Shull, F. (2013), Comparing four approaches for technical debt identification, Software Quality Journal, 1-24.

Zazworka, N.; Seaman, C. & Shull, F. (2011), Prioritizing design debt investment opportunities, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt.

Zazworka, N.; Shaw, M. A.; Shull, F. & Seaman, C. (2011), Investigating the impact of design debt on software quality, MTD 11: Proceedings of the 2nd Workshop on Managing Technical Debt. New York, NY, USA: ACM, 2011, pp. 17–23. [Online]. Available: http://doi.acm.org/- 10.1145/1985362.1985366

Zazworka, N.; Spinola, R. O.; Vetro, A.; Shull, F. & Seaman, C. (2013), A case study on effectively identifying technical debt, EASE 13: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering.

Zazworka, N.; Vetró, A.; Izurieta, C.; Wong, S.; Cai, Y.; Seaman, C. & Shull, F. (2014), Comparing four approaches for technical debt identification, Software Quality Journal 22(3), 403-426.

References

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.

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P. 2015. The financial aspect of managing technical debt: A systematic literature review, Information and Software Technology, Volume 64, April 2015, Pages 52-73, ISSN 0950-5849, http://dx.doi.org/10.1016/j.infsof.2015.04.001.

Basili, V. R., Shull, F., Lanubile, F. (1999), Building Knowledge Through Families of Experiments. IEEE Transactions on Software Engineering, Vol. 25, No. 4, July/August.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software 80, 571–583.

Budegn, D., Turner, M., Brereton, P., Kitchenham, B. (2008), Using Mapping Studies

in Software Engineering. In the Proceedings of PPIG Psychology of Programming Interest Group, Lancaster University, UK, pp. 195–204.

Cai Y., Kazman R., Jaspan C., Aldrich J. (2013), Introducing Tool-Supported Architecture Review into Software Design Education, on IEEE 26th Conference on Software Engineering Education and Training (CSEE&T), vol., no., pp.70,79, 19-21.

Cunnigham, W. (1992), The WyCash Portfolio Management System, in Addendum to the proceedings on Object-oriented programming systems, languages, and applications, pp. 29-30.

Diehl, S. (2007), Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer-Verlag New York, Inc.

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

Kitchenham, B., Dybå, T., Jorgensen, M. (2004) Evidence-based Software Engineering, Proceedings of the 26th ICSE, Scotland, UK.

Kitchenham, B., Charters, S. (2007), Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Kitchenham, B., Mendes, E., Travassos G.H., (2007a), Cross versus within-company cost estimation studies: a systematic review, IEEE Transactions on Software Engineering 33 (5) (2007) 316–329.

Li, Z., Avgeriou, P., Liang, P. (2015). A systematic mapping study on technical debt and its management. In Journal of Systems and Software, Volume 101, March 2015, Pages 193–220. doi:10.1016/j.jss.2014.12.027

Novais R. L., Torres A., Mendes, T. S., Mendonca M., and Zazworka N. (2013), Software evolution visualization: A systematic mapping study, IST, 55(11):1860 – 1883.

Nunamaker, Jr., Jay F. and Chen, Minder and Purdin, Titus D. M. (1990) Systems Development in Information Systems Research, J. Manage. Inf. Syst., Armonk, NY, USA, pp. 89-106.

Petersen, K., Feldt, R., Mujtaba, S., Mattson, M. (2008), Systematic mapping studies in software engineering. In the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy.

Petticrew, M. & Roberts, H. (2006), Systematic Reviews in the Social Sciences: A Practical Guide, Blackwell Publishing.

PFleeger, S. (2005) Software Engineering: Theory and Practice. Third Edition. Prentice Hall.

Pressman, R. S. (1997), Software Engineering: A Practitioner's Approach, McGraw-Hill, pp. 253-259

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

Tomas, P., Escalona, M.J., Mejias, M., (2013), Open source tools for measuring the Internal Quality of Java software products. A survey, Computer Standards & Interfaces, vol 36, Issue 1, pp. 244-255.

Villar, A., Matalonga, S. (2013), Definiciones y tendencia de deuda técnica: Un mapeo sistemático de la literatura. Anais do CIBSE13 - Congresso Ibero-Americano em Engenharia de Software, Montevideo, Uruguai, Abril 8, 9 e 10, 2013, pp 33-46.

Wohlin, C. Runeson, P., Host, M., Ohlsson, M., Regnerll, B., Wesslén, A. (2000), Experimentation in Software Engineering: an introduction, Kluwe Academic Publishers, USA.