HEARING THE VOICE OF SOFTWARE ARCHITECTS ON PAYMENT-RELATED PRACTICES OF TECHNICAL DEBT IN COLOMBIA

ESCUCHANDO LA OPINIÓN DE LOS ARQUITECTOS DE SOFTWARE SOBRE LAS PRÁCTICAS DE PAGO DE LA DEUDA TÉCNICA EN COLOMBIA

PhD. Darío Correal[2], MSc. Boris Pérez[1,2], MSc. Camilo Castellanos[2]

[1] **Universidad Francisco de Paula Santander,** Facultad de Ingeniería, Grupo de Investigación en Inteligencia Artificial (GIA)
E-mail: {borisperezg}@ufps.edu.co
[2] **Universidad de los Andes**, Facultad de Ingeniería, Grupo de Investigación en Tecnologías de Información y Construcción de Software (TICSw)
E-mail: {dcorreal, br.perez41, cc.castellanos87}@uniandes.edu.co

Abstract: Technical debt (TD) describes technical decisions that can give the company a benefit in the short term but possibly hurting the overall quality of the software in the long term. Architectural decisions are considered one of the most common sources of TD, therefore, it becomes relevant to understand what practices related to TD payment are considered by software architects in comparison with engineers and managers. To this end, we used a survey research method to collect and analyze a corpus of 28 software architects from Colombia. Results showed that refactoring is the most cited practice by all three groups, being the most cited practice by software architects (42.9%). Additionally, practices related to prevention and creation of a favorable setting as part of TD payment initiatives were cited. In the end, everything leads to the source code, then becoming the first place to pay the debt.

Keywords: technical debt, payment practices, software architecture, insightd

Resumen: La deuda técnica (DT) describe las decisiones técnicas que pueden beneficiar a la empresa a corto plazo pero que posiblemente perjudiquen la calidad del software a largo plazo. Las decisiones de arquitectura son la principal fuente TD, por lo tanto, se hace relevante entender qué prácticas asociadas al pago de la TD son consideradas por los arquitectos de software en comparación con los ingenieros y gerentes. Para esto, se utilizó una encuesta que permitió recoger y analizar un corpus de 28 arquitectos de software de Colombia. Los resultados mostraron que la refactorización (refactoring) es la práctica más citada por los tres grupos, siendo la más citada por los arquitectos de software (42.9%). Además, se citaron prácticas relacionadas con la prevención y la creación de un entorno favorable para pagar la TD. Al final, todo lleva al código fuente, convirtiéndose en el primer lugar para pagar la deuda.

Palabras clave: deuda técnica, prácticas de pago, arquitectura de software, insightd.

1. INTRODUCION

Software companies usually have to deal with tight schedules and deadlines to release software in faster cycles, and therefor, increasing the pressure for the development teams (Yli-Huumo *et al.*, 2016). Technical debt (TD) represents technical decisions that can give the company a benefit in the short term (Kruchten *et al.*, 2012; Verdecchia, 2018) but possibly hurting the overall quality of the software and the productivity of the development team in the long

term. Among these technical decisions, architectural decisions are the most important source of TD (Ernst *et al.*, 2015). Therefore, it becomes crucial to understand how TD is perceived by software architects, in terms of practices to deal with TD.

Despite its relevance, there is still a lack of empirical evidence about TD payment-related practices used by software architects in real-life software development teams (Li *et al.*, 2016; Rios *et al.*, 2018a; Seaman and Guo, 2011; Power, 2013).

This study focuses on the acknowledgment of the practices used on TD payment from the point of view of software architects in real-life software systems projects in Colombia. To achieve this, we performed an industrial survey with 28 software architects from 132 software practitioners in Colombia. These answers were compared against answers from management roles (project manager, business analyst, etc) and engineering roles (developer, tester, etc). The contributions of this work are two-fold. First, this study presents a list of the TD payment-related practices (refactoring being the most cited). And second, a numerical comparison of similarity between the list of practices cited by software architects against management and engineer groups.

Software practitioners can benefit from this proposal to support the selection of strategies to keep their software systems healthy, thought the improvement of the existing processes and tools.

The rest of the paper is organized as follows: in Section 2 we present a description of the InsighTD project history. In Section 3, we present the survey design, whose results are presented in Section 4. Implications for researchers and practitioners are presented in Section 5. Section 6 presents the comparison to previous work. Finally, in Section 7, we present threats to validity, and in Section 8 we conclude the paper.

2. INSIGHTD PROJECT

InsighTD is a globally distributed family of industrial surveys initiated in 2017 and planned cooperatively among Technical Debt (TD) researchers from around the world. Main goal of this project is to organize an open and generalizable set of empirical data on the state of practice in the TD area. To date, researchers from Brazil, Chile, Colombia, Costa Rica, Finland, India, Italy, Norway, Saudi Arabia, Serbia, and the United States have joined the project.

Rios (Rios *et al.*, 2018b) discussed the basic survey design and the preliminary results of the first round of InsighTD, and complemented this discussion, focusing specifically on the causes and effects of TD in agile software projects. Pérez (Pérez *et al.*, 2019) focused on how practitioners react to the presence of debt in the Chilean software industry. More recently, Freire (Freire *et al.*, 2020) investigated preventive actions that can be used to curb the occurrence of TD and the impediments that hamper the use of those actions.

Thus, although significant analysis has already been conducted over the available InsighTD data, much still remains to be studied. In particular, a noticeably absent and important perspective is the one from the architect's point of view.

3. METHODOLOGY

This research was designed with the goal of characterizing comprehensively the current state of practices related to TD payment. Based on our research goal, we derived the following two research questions:

RQ1: From a software architect's point of view, what are the practices related to TD payment used by software development teams?

RQ2: Is there any difference of TD payment-related practices among software architects, engineers and managers?

Data gathering was done using Google Forms. This tool allowed us to increase the number of possible participants. Invitations were sent by email to software practitioners and the survey was anonymous. Survey questions were defined within the InsighTD replication package and was made up of 28 questions, previously described in (Rios *et al.*, 2018b).

Demographics questions (Q1 to Q8) ask participants about, for example, the size of his/her company, size of the system (in terms of LOC) he/she is working on, number of people involved in that project, participant's role, and her/his level of experience in that role. Questions Q9 to Q15 seek information about how familiar the respondent is with the TD concept. Questions Q16 to Q19 support the identification of the causes that lead development teams to insert debt items into their projects. Questions Q20 and Q21 look to identify effects of the presence of TD in software projects. Finally, Questions Q22 to Q28, were used to provide an understanding on how TD has been managed in practice, in particular with respect to prevention, repayment,

and monitoring. The full questionnaire was previously presented in (Rios *et al.*, 2018b). In the context of this work, we considered for analysis the characterization (Q1-Q8) and payment practices of TD (Q26 and Q27).

Questionnaire validation included three steps: an internal validation, an external validation, and a pilot study (Rios *et al.*, 2018b). To reach the target population (software practitioners) we utilized the social media platform LinkedIn. LinkedIn gave us direct access to a large number of professionals with whom we did not have previous contact.

The survey instrument is composed of a mix of closed and open questions. For closed-ended questions, we used descriptive statistics to get a better understanding of the data. Answers for open-ended questions were codified using a code schema provided with the InsighTD replication package. We initially applied manual open coding resulting in a set of codes. The process was performed iteratively revising and unifying codes at each cycle of analysis until reaching the state of saturation, i.e., a point where no new codes were identified.

Data analysis was done focusing on architects and comparing its results against management and engineer groups. We are aware that software architects could be part of the engineer group, however, considering the focus of this study, it was decided to have software architects as a distinct group.

4. RESULTS

In total, 132 practitioners answered the survey. After filtering answers according to their role, we found 28 (21.2%) participants classified as software architects, 37 (28%) as managers and 67 (50.8%) as engineers, as presented in Fig. 1.

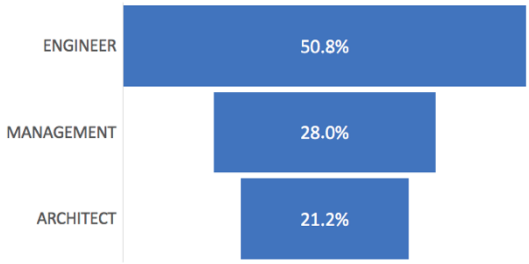

Fig. 1. Practitioners distribution by role

Participants are well distributed among small (28.6%), medium (46.4%), and large (25%) companies. Related to the size of development teams, most (28.6%) reported working in teams of 5-9 people and teams of 10-20 people (28.6%). Regarding the age of the system developed in the project, most indicated age 1 to 2 years (46.4%). There are also a significant number of systems represented from 2 to 5 years (21.4%). Most respondents identified themselves as proficient (39.3%), followed by expert (28.6%), and competent (21.4%). In general, the questionnaire was answered by professionals with experience in their functions.

4.1 Main Practices Related to TD Payment (RQ1)

Fig. 2 presents the most commonly cited practices used and related to TD payment used by development teams from the point of view of software architects (Questions 26 and 27). The first two practices correspond to 62% of the set of all practices.

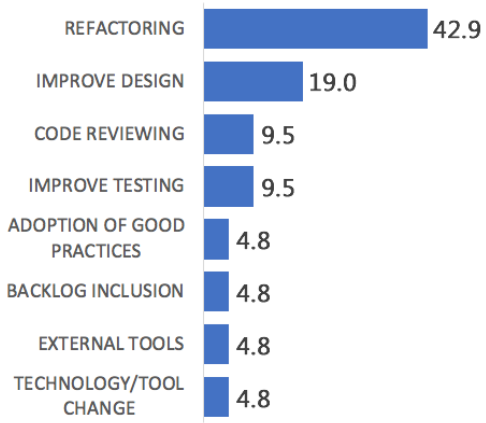

Fig. 2. Practices related to TD payment

From Fig. 2, we can observe that “refactoring” is the most cited practice used for TD payment (9 citations) form the point of view of the software architects. Refactoring consists of performing a series of small behavior-preserving transformations to improve an existing code, design or architecture of a software system (Li *et al.*, 2015). This practice is two times greater than the second most cited practice: “Improve design” (4 citations). According to Ernst (Ernst *et al.*, 2015), architecture debt is the most common source of technical debt, so it is expected to have this practice (improve design) at least in the first three positions. “Code reviewing” is in third place followed by “improve testing”. These two last practices could be considered good practices. Following activities are: “Adoption of good practices”, “backlog inclusion”, “external tools” and “technology/tool chance”.

By going further into the analysis of the whole set of practices, we realize that some practices do not allow the elimination of TD items. For instance, the practice “adoption of good practices” contributes to create a favorable

scenario for eliminating TD items but does not eliminate the item by itself. Other practices such as "improve testing" can be seen as preventative practices. Thus, TD payment-related practices encompass practices associated with TD payment, prevention, and also the creation of a favorable scenario for paying off debt items.

All of these practices can be considered as technical issues; therefore, technical issues are important practices related to TD payment for software architects.

4.2 Comparison of TD payment practices among Software Architects, Engineers and Managers (RQ2)

In this work, we investigate how different or similar these practices are (perceived by software architects) in comparison with the other two groups: engineering and management. In Fig. 3, the top four practices by role are presented.

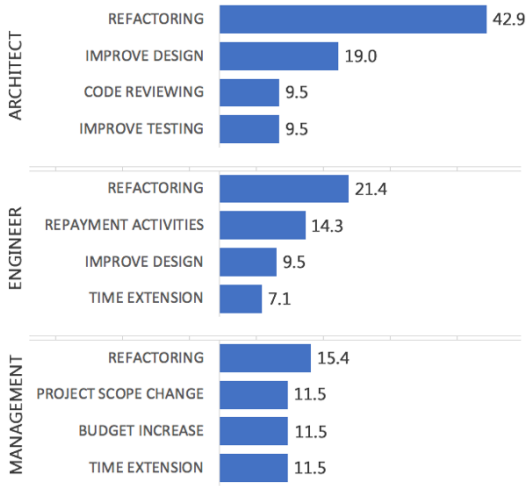

Fig. 3. Top four practices related to TD payment in all roles

Results showed that all three groups have "refactoring" as the most cited practice. This is the only similarity among the groups. Software architect and engineer groups also share "improve design" practice. Engineer and management groups share "time extension" with similar percentage. Activities for management group look more focused in the management aspect of the software development process. In the other side, architect and engineer groups are more focused in the technical aspect of the software development process.

Fig. 3 support the understanding of differences by doing a visual comparison. However, there is a necessity to improve this comparison by measuring quantitatively how similar the TD payment practices among the three groups are. We used a similarity measure for indefinite rankings called Rank-Biased Overlap – RBO (Webber *et al.*, 2010). RBO is defined as follows:

$$RBO(S,T,p) = (1-p)\sum_{d=1}^{\infty} p^{d-1} \cdot A_d \qquad (1)$$

where S and T are the ranked lists; p is the probability of looking for overlap at depth $d+1$ after having examined element at d. The smaller the p value, the more top-weighted the metric. A_d is the agreement between S and T at depth d, i.e. the proportion of S and T that are overlapped. Fig. 4 depicts the RBO comparison among the list of TD practices of the three groups (one line for each pair of groups). By increasing the p value, this comparison aims to explore how similar these practices are per group at the top of their rankings and at the bottom of their rankings. Comparison went from $p=0.5$ (top 2 elements approx.) to $p=0.97$ (top 33 elements approx.).

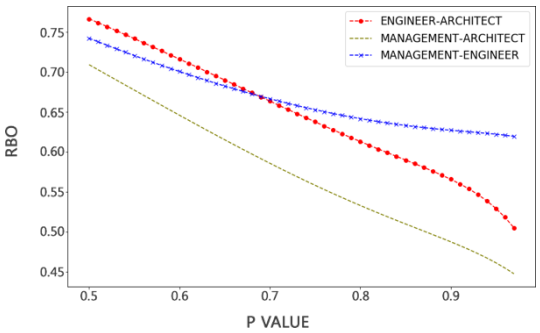

Fig. 4. RBO of TD payment-related practices

For $p=0.5$, the Engineer-Architect pair showed the highest similarity (RBO=0.77), but at $p=0.97$, the RBO of this pair decreased to 0.5. This means that the more practices are compared, the less similar they are. This could be explained considering the number of items of both lists. Also, this behavior would be more or less expected when comparing lists. On the other hand, the Management-Architect pair showed the lowest similarity at $p=0.5$ and $p=0.97$. This means that payment-related practices described by these two lists are different in comparison with the other two pairs of lists. The last pair, Management-Engineer ended being the most similar pair among the others. At p=0.97, the RBO was 0.62. This result doesn't imply that management and engineer roles are the most similar of all, but only that, in the case of this study, this pair has more similarities in the TD payment-related practices.

5. DISCUSSION

Section 4 presented "refactoring" as two times greater than the second most cited practice,

"improve design". Refactoring is a technique for improving the design of an existing code base. In the vast amount of answers, refactoring was used by software teams in terms of this definition, in other words, code refactoring. Refactoring of designs or architectures was coded under other practices such as: "Improve design" or "architectural changes".

There are payment-related practices that would require a deeper analysis, for example: "Code reviewing". This is not a practice to pay off the debt, but it is a practice that is more likely to contribute to create a favorable scenario for eliminating TD items, and it does not eliminate the item by itself. Other examples of practices are: "Budget increase" and "time extension". More time could mean injecting more debt while fixing the code. And also, the team would need more time to fix some issues along with the debt. Increasing the budget of the project will also mean that the software team would have to include new functionalities and not only pay off the debt. No client will pay for fixing code issues that the user will not see. It is important to remark that TD consequences are related to maintainability and evolvability.

These payment practices by themselves are not enough. It is necessary to analyze the differences among them in order to understand the nature of the required changes (improvements) and the resources needed. For example, the frequency of the payment practice: a software team can either choose to pay TD continuously, occasionally, or not at all (Yli-Huumo *et al.*, 2016). Martini and Bosch (Martini *et al.*, 2015), suggest that TD should be payed using partial repayments. In this way, the risk associated with the payment of the debt can be minimized.

Another aspect that needs to be considered is the cost related to the debt. How much would it cost to pay off the debt? vs. How much does it cost to maintain the debt? According to Martini and Bosch (Martini *et al.*, 2015), it could be more profitable to delay the refactoring, i.e. continue paying interest.

As can be seen, it is not just about the payment of the debt, it is also about the analysis of several aspects before any decision are made.

6.1Implications to researchers and practitioners

Software practitioners can benefit from the results of this study by using the list of the most cited practices related to TD payment used in the Colombian industry as a guide to support initial efforts to understand their debt and to pay it off from their software projects.

For researchers, our results support future research by providing insights into software practitioners' perspectives on causes leading to TD occurrence and practices related to TD payment. Finally, the global family of surveys not only allows researchers to reproduce the results and their interpretation, but also allows practitioners to evaluate their own TD situation against overall industrial trends.

6. RELATED WORK

Although there are previous works that point out some causes and effects of TD (Yli-Huumo *et al.*, 2014; Ernst *et al.*, 2015), their sample sizes tend to be quite small, or limited to a small number of organizations. Besides, a predefined list of TD causes constrained participants to fit their experience into that list. Further, half of the relevant studies focused on architectural TD, just one type of debt, and payment related practices have not been sufficiently studied.

The survey presented in (Ernst *et al.*, 2015) reports the results obtained from 1,831 software practitioners. They studied the common understanding of the TD term, how much of technical debt is architectural, and the management practices and tools used to deal with TD. In (Yli-Huumo *et al.*, 2014), Yli-Huumo investigated the causes and effects of TD by interviewing 12 persons in a Finnish software company. Their results reported that TD is mainly the result of intentional decisions to reach deadlines, and customer satisfaction was the main reason for taking TD in the short-term, but it turned to economic and quality issues in the long-term.

Related to InsighTD, although initial analysis has already been conducted over the available data, much still remains to be studied. In particular, the data has yet to be analyzed with regards to how TD is perceived by architects, and how this perception change among other groups. Pacheco (Pacheco *et al.*, 2019) reported another InsighTD replication in Costa Rica, where 156 software professionals reported that TD was the product of preventable situations, TD was monitored for slightly more than half of the cases, and TD was not paid in most cases.

To summarize, this work becomes a part of the InsighTD project, but, unlike the previous replications, we aim to study the TD causes and practices related to TD payment from the point of view of software architects, and how these causes and payment practices can be associated.

7. THREATS TO VALIDITY

There are threats to validity in this work that we attempt to mitigate and remove entirely when possible. First, regarding construct validity, to prevent hypothesis guessing and evaluation apprehension (Wohlin *et al.*, 2012), we explained in the invitation to the survey the goal of the study and request that interviewees reply to questions by relying on their own background. Second, regarding conclusion validity, to avoid potential coding process dependencies on the researcher's subjective criteria, the coding activity was performed individually by two researchers, and then, discussed until an agreement was reached.

Maturation is the main threat to internal validity of this study. It implies that the participants can react differently as time passes, in this case, if the survey is too long (Wohlin *et al.*, 2012). The fact that all participants answered the whole questionnaire is a signal of that this threat was not raised. Finally, regarding external validity, although the results cannot be generalized, the population provides representative results from the perspective of the software industry.

8. CONCLUSIONS

The contributions of this work are two-fold. First, we presented a list of the practices (refactoring being the most cited) related to TD payment. And second, a numerical comparison of similarity between the list of practices cited by software architects against management and engineer groups were presented. These contributions were done from the point of view of 28 software architects from Colombia. Results were compared against answers from engineering and management groups in order to improve the understanding of the insights.

We found that “refactoring” is the main practice used to pay off the debt, followed by “improve design” and “code reviewing”. Refactoring was also the main practice in all three groups. This could be the only similarity among the three groups, having software architects and engineer the most similar list of practices.

Activities for management group were more focused in the management aspect of the software development process. In the other side, architect and engineer groups were more focused in the technical aspect of the software development process.

The next steps of this research include: (i) a deeper analysis (including demographics variables) to identify possible patterns of TD payment related practices, (ii) investigation of how or if types of debt influence them, (iii) running other possible analyses including others reactions to TD, such as monitoring practices and preventative actions. We have also yet to include the replication data from other countries such as: Finland, Saudi Arabia, Serbia, and Costa Rica.

REFERENCES

Ernst N., Bellomo S., Ozkaya I., Nord R., and Gorton I. (2015). Measure it? Manage it? Ignore it? software practitioners and technical debt. In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. Association for Computing Machinery, New York, NY, pp- 50–60.

Freire S., Mendonça M., Falessi D., Seaman C., Izurieta C. and Spínola R. (2020). Actions and impediments for technical debt prevention: Results from a global family of industrial surveys. To appear in proceedings of the 35th ACM/SIGAPP Symposium On Applied Computing.

Kruchten P., Nord R., and Ozkaya I. (2012). Technical Debt: From Metaphor to Theory and Practice. IEEE Software, Vol. 29, No. 6.

Li Z., Avgeriou P. and Peng L. (2015). A systematic mapping study on technical debt and its management. Journal of Systems and Software, Vol. 101, pp. 193-220.

Martini A., Bosch J. and Chaudron M. (2015). Investigating architectural technical debt accumulation and refactoring over time. Information Software Technology, Vol. 67, No. C.

Pacheco A., Marín-Raventós G. and López G. (2019). Technical Debt in Costa Rica: An InsighTD Survey Replication. In proceedings of the International Conference on Product-Focused Software Process Improvement, pp. 236-243.

Pérez B., Brito J, Astudillo H, Correal D, Rios N., Spínola R., Mendonça M. and Seaman C. (2019). Familiarity, causes and reactions of software practitioners to the presence of technical debt: A replicated study in the chilean software industry. In proceedings of the 38th International Conference of the Chilean Computer Science Society, pp.1-7.

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 Proceedings of the 4th International Workshop on Managing Technical Debt (MTD ’13). IEEE Press, 28–31.

Rios N., Mendonça M. and Spínola R. (2018a). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology, Vol. 102, pp. 117-145.

Rios N., Spínola R., Mendonça M. and Seaman C. (2018b). The Most Common Causes and Effects of Technical Debt: First Results from a Global Family of Industrial Surveys. In proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, No. 39, pp. 1-10.

Seaman C., Guo Y. (2011). Chapter 2 - Measuring and Monitoring Technical Debt. Advances in Computers, Vol. 82, pp. 25-46.

Verdecchia R. (2018). Architectural Technical Debt Identification: Moving Forward. In proceedings of the IEEE International Conference on Software Architecture Companion (ICSA-C), Seattle, WA, pp. 43-44.

Webber W., Moffat A. and Zobel J. (2010). A Similarity Measure for Indefinite Rankings. ACM Trans. Inf. Syst, Vol. 28, No. 4.

Wohlin C., Runeson P., Hst M., Ohlsson M., Regnell B. and Wessln A. (2012). Experimentation in Software Engineering. Springer Publishing Company, Incorporated.

Yli-Huumo J., Maglyas A., and 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 proceedings of the International Conference on Product-Focused Software Process Improvement.

Yli-Huumo J., Maglyas A., and Smolander K. (2016). How do software development teams manage technical debt? - An empirical study. Journal of Systems and Software, Vol. 120, No. C