Software Architecture Social Debt: Managing the Incommunicability Factor

Damian A. Tamburri

Abstract—Architectural technical debt is the additional project cost connected to technical issues nested in software architectures. Similarly, many practitioners have already experienced that there exists within software architectures a form of social debt, that is, the additional project cost connected to sociotechnical and organizational issues evident in or related to software architectures. This paper illustrates four recurrent antipatterns or community smells connected to such architectural social debt and outlines a means to measure the additional project cost connected to their underlying cause: decision incommunicability. Evaluating the results in multiple focus groups, this paper concludes that studying social debt and community smells at the architecture level may prove vital to rid software development communities of critical organizational flaws incurring considerable additional cost.

Index Terms—Social debt, social debt cost estimation, social debt in software architecting, technical debt.

I. INTRODUCTION

In LAYMAN’S terms, social debt [52] is the additional project cost in the current state of affairs caused by sub-optimal sociotechnical decisions, e.g., decisions that result in breaking sociotechnical congruence [8]. The additional cost may be connected to a wide variety of possible decisions, e.g., changing the organizational structure [33] of the development network [27] (e.g., through outsourcing), changing the development process (e.g., by adopting agile methods), and leveraging on (too big or too small) global teams. This paper reports on a case study of social debt in a context where software architecture is subjected to community antipatterns, better known as smells [52], during the process of architecting, much similar to the observations of Martini and Bosch [24], [25] who report nasty organizational causes and effects leading software architectures to accumulate technical debt in the long run.

Previous work [49] already highlighted the recurring organizational antipatterns or community smells in software architectures and the process of architecting—such smells are precursors to social debt much like code smells that are precursors to technical debt [19]. The smells, in question, all reflect a lack of architecture decision communicability across the software development network [27]. For example, architecture decisions can take place “by osmosis,” that is, considering information that crosses every possible communication link across the development and operations network. In this circumstance, loss of essential information is almost inevitable. For example, observed architecture decisions in this circumstance emerged from the following sequence of events: 1) clients encountered difficulties; 2) these difficulties were communicated to operators; 3) operators discussed and alerted developers; and 4) developers forced architects into architectural change, sometimes making partial architecture decisions and code refactoring of their own. The problem is that much information, rationale and requirements, can go lost in translation as change requests travel from client to architects. Several nasty circumstances emerge as linked to this chain of events, such as architecture erosion [40], poor architecture documentation, lack of vision, and, eventually, mistrust. These circumstances lead to an additional project cost from both a social (e.g., lack of vision) and a technical perspective (e.g., architecture erosion)—the key commonality between these aspects is software architecture incommunicability, that is, the inability to directly communicate architecture decisions to their respective targets (e.g., owners of the affected components). Much around software architecture incommunicability is still unknown and, what is more, previous work [49] merely provides a rudimentary social debt measurement framework called “Debt-Aimed archItecture-Level Incommunicability Analysis” (DAHLIA) to elicit some of the incommunicability social debt for further analysis.

With respect to previous work [49], which partially addressed architecture-level community smells and their basic assessment, this paper fleshes out the full length of details under architecture incommunicability as a social-networks’ analysis (SNA) measurement unit [28] and outlines the theoretical underpinnings of DAHLIA, the social debt management framework designed to measure and quantify the debt. More specifically, the following novel contributions are presented.

1) In previous work, only two out of four architecture-level community smells were reported; this extended version fleshes out full details of the remaining two and operates a theoretical synthesis of their root cause, to identify the architecture incommunicability factor—the definition of the factor as an SNA measurement unit is completely novel (see Section IV).

2) In previous work, the DAHLIA framework was theorized and fleshed out as a new idea; this extended...
version of this paper provides full details over the theoretical and software engineering economics underpinnings behind the framework—furthermore, the application of DAHLIA in the context of an industrial case study is completely novel (see Section V).

3) Previous work did not evaluate further the results using other industrial studies; this extended version provides further evaluation of all above-mentioned results using two structured industrial focus groups (see Section VI).

The results, in this paper, stem from a longitudinal, real-life industrial case study and were confirmed using industrial focus groups.

More specifically, to confirm the validity of the architectural social debt community smells and their measurement framework reported in this paper, the connected notions were stressed in two industry–academy mixed focus groups as part of a Dagstuhl Seminar on Technical Debt.\(^1\) The focus groups counted a total of nine senior members from several top industrial players, including Google, CAST Software, HP Inc., and VNomic. As part of the results of the focus groups, the validity of the architectural social debt community smells reported in this paper along with the DAHLIA framework to manage them was confirmed. Furthermore, the involved practitioners allowed a deeper exploration of previously unknown dimensions over the social debt phenomenon. For example, the definition and ramifications of social debt as a metaphor were refined with an even deeper relation to technical debt—a form of sociotechnical debt, wherefore the social causes always lead to nasty technical phenomena as well (e.g., code churn, unwanted changes, replicated code, and more). This latter form of sociotechnical debt reflects additional project cost which is caused by suboptimal sociotechnical decisions but manifests as negative and invisible characteristics in source code or connected software artifacts (e.g., requirements, software architecture, documentation, and so on) rather than delays or communication friction across the organizational structure.

This paper concludes by arguing that the joint study of social and technical debt should be further encouraged and structured, possibly by combining efficient community smells detection mechanisms with technical debt measurement as well, e.g., in a mixed-methods research agenda.

Structure of This Paper: The rest of this paper is structured as follows. Sections II and III outline the research design, materials, and methods. Section IV fleshes out study results, elaborating on research questions (RQs) individually. Section VII evaluates the results. Section VIII outlines the related work. Finally, Section IX concludes this paper.

II. RESEARCH DESIGN

A. Research Objective and Questions

The research objective, this research set out to explore in the scope of the material reported in this paper, is finding out whether there exist recurrent software architecting circumstances that create additional project cost from a sociotechnical and/or organizational perspective. For example, whether there are phases in the processes of evaluating, coming up with, or disseminating architectural decisions, wherefore social or organizational friction may be causing additional project cost or unnecessary organizational strain. The theory that the research endeavor set out to confirm is captured in previous work [52] and essentially relates to the existence of social debt connected to recurrent sociotechnical antipatterns known as community smells—in the previous social debt work, several such community smells were encountered. In the context of this paper, the object of the study was to understand whether the industrial case at hand may contain such smells and to what degree their effect manifests.

To flesh out the above-mentioned objective, three main RQs were formulated.

RQ1: Are there any community smells emerging in the process of software architecting?

This RQ inherits from the notion of community smells, that is, recurrent sociotechnical or organizational antipatterns connected to suboptimal organizational structures. We encountered several community smells already in the previous industrial case-study research [50], [52], [53]. Stemming from the notion in the question, the research design, in this paper, is aimed at elaborating further on what circumstances actually create architectural social debt—results for this RQ are reported in Section IV.

RQ2: Provided that architectural community smells do exist, can the extent of the impact for reported community smells be measured?

This RQ aims at quantifying, if possible, the architectural social debt associated with the discovered community smells. Because the underlying cause for all smells found is the newly found incommunicability factor, the response to this question focuses on fleshing out incommunicability and its measurement—results for this RQ are reported in Section V.

RQ3: Can social debt be addressed and mitigated?

This RQ aims at understanding whether software architects or practitioners involved in the study perceived the architectural social debt in their scenario and employed mechanisms or organizational procedures to address it—results for this RQ are reported in Section VI.

B. Unit of Analysis

The results, in this paper, are based on a study in a large IT service provider (called Capita from now on) for the aviation industry. Capita has around 3000 employees in several locations in Germany and around Europe. Also, Capita controls several offices in 14 other countries.

The context of investigation is a software project (called Integra, from now on) featuring the integration of two very different software products (called RED and GREEN, from now on).

The organizational scenario investigated in Integra is shown in Fig. 1,\(^2\) Nodes in the graph are people part of the development network under study, while edges represent the existence of reported frequent interactions in-between network

\(^1\)https://tinyurl.com/o8d7yuz

\(^2\)Names are unavailable to preserve privacy and uphold nondisclosure agreements.
Fig. 1. Organizational scenario under investigation of the Integra project.

members. In summary, the community of developers under analysis involves two geographically distributed production sites A (headquarters) and B (remote site) providing developers as well as customer support and operations. Sites A and B are both responsible for the implementation of incoming user requests (e.g., new requirements, revised requirements, bug reports, and so on), but site B also handles the maintenance of both products being integrated.

A big organizational difference between sites A and B is that responsibilities in the remote site B are limited to follow what is decided by product managers in site A. Product managers and architects in site A are responsible for the management, software architecture, requirement elicitation, and critical decision-making. In addition, while RED is a well-established product, active for well over 10 years, and managed following an established waterfall model, the GREEN product (and people) are relatively new and adopt agile methods. The RED + GREEN integration side resides mainly in site A and acts as an intermediary between sites A and B. RED + GREEN site is intended to integrate decisions and their implementation on both products into one coherent whole, sometimes lending expertise from either RED or GREEN. Essentially, the scope of people working on RED is to maintain critical and unreplicable operations for the product while aiding its integration with GREEN.

Conversely, the scope of people working on GREEN is to adapt some of its interfaces to interact with RED while developing new functionality. Finally, the scope of people working on the RED + GREEN integration is to integrate RED and GREEN interfaces using middleware, wrappers, and similar integration technology.

III. RESEARCH MATERIALS AND METHODS

To obtain the results in this paper, real-life case-study research was employed, namely, the research aimed at studying the social debt phenomenon and, more specifically, the sociotechnical circumstances leading to software architecting social debt investigating contemporary phenomena connected to the research target in their specific context; in this case, the context reflects a specific and single unit of analysis, namely, a large, globally distributed software renewal and integration project featuring 13 individuals arranged across a heterogeneous set of roles. The reporting of the case study follows the guidelines of Runeson and Höst [44], wherefore case-study research is aimed at adopting surveys, controlled experiments, and/or action research to study the phenomenon in question.

A. Data Set

For the case reported in this paper, the following data elicitation and sources were adopted.

1) A total number of two interview survey rounds³ (with an average of 90 min per interview) involving all practitioners previously captured in Fig. 1. Practitioners involved in the case study were questioned at the start of the study with a series of open questions aimed at eliciting data over the organizational and sociotechnical phenomena taking place in their organizational scenario; due to the exploratory nature of the study, any explicit focus over software architecture, community smells, social debt, and other notions in the original social debt theory were purposefully avoided [52]. The concluding round of interviews was aimed at establishing the resonance (i.e., the recurrent and still perceivable presence of the indications and evidence from the elicited data points) of all observations made over the eight-month length of the study.

2) Study of a total number of 14 archival data items for a total of 182 pages.⁴ An associate of the author of this paper embedded in the research context to lend software measurement and data integration expertise. The study of the archival data was aimed at supporting the possible

³A summary of key interviews is available online: https://tinyurl.com/interviews-TCSS.

⁴These documents are protected by nondisclosure agreements and cannot be disseminated further.
1. Label certain observable phenomena
2. Discover categories across phenomena
3. Further elaborate categories (properties, inheritance, etc.)
   1. Compare categories, discover patterns
   2. Elaborate on patterns and relate categories = core-categories
      1. Compare/relate 6C models
      2. Sort core categories, e.g., find precedence
      3. Exhaust model: all memos fit into theory

reapplication of social debt and community smells’ models from previous research [52] using descriptive data already available from the new industrial context.

With the above-mentioned connotation, the study assumes the form of an observational, interpretative, en-field study. Interviews\(^5\) were structured according to procedures and guidelines suggested by Neville-Neil [34]. It should be noted that social debt itself was never mentioned during data elicitation to avoid bias. In addition, focus groups were instrumented according to the guidelines previously introduced by Morgan and Krueger [32]. Finally, the evaluation focus groups were structured according to the “Working Group” organizational and social structure in [55]. The aim of the focus groups was to provide the validation of observations made and refinement where possible, e.g., for unclear or misunderstood concepts. Following strict nondisclosure agreements, all transcriptions were completely anonymized at the source.

B. Analysis Methods

Finally, concerning data analysis, a grounded-theory (GT)-based approach was adopted, wherefore all of the interview data and action-research field notes were analyzed by two observers and interrater reliability (IRR) established (see Section III-B.3 for a discussion of observer triangulation). GT [10], [45] is a well-established technique in the field of software engineering and features the constant labeling and comparison of the data set against a preestablished or emergent set of categories.

1) Grounded-Theory Specifics: Fig. 2 (tailored from previous work [39], [54]) shows the series of steps to be applied sequentially and iteratively as part of a GT-based analysis. Essentially, observers of a phenomenon are required to label any piece of evidence (in our case, source code elements) with a label, indicating the meaning of that piece of evidence (e.g., a method call, an entire method, and an entire Java Class, depending on the desired accuracy—a class could be labeled with class purpose or functional role). These labels can then be analyzed and clustered according to the emergent categorization. Subsequently, codes can be merged, e.g., according to semantic similarity principles [42]. As labels (or better codes) are merged, categories become enriched with further details (e.g., some labels likely become properties of some others and so on). Moreover, in the shift from open to selective coding, patterns are discovered across categories (e.g., recurrent categories of sections of source code may denote a type of software component). The goal of selective coding is to relate categories together until there are no other additional possible relations

\(^5\)An interview guide is available in the related Dagsstuhl proceedings [1].
to be discovered. The GT approach previously outlined was employed in the context of this paper and was tailored from previous work [52]. In the following, more details are given as to how the above-mentioned process was instantiated.

1) Open Coding (4 Phases):
   a) Pilot Study: A set of three interviews were randomly selected to generate an initial set of codes by an independent researcher.
   b) Develop Initial Theory: Based on the pilot study, an initial theory was generated.
   c) Constant Comparison: The pilot study generated an initial set of 39 codes. These were organized into a hierarchy of codes based on the emerging relations between the concepts. Thus structured, the start-up list of codes was used to code the rest of the interviews. Each interview transcript was analyzed line by line with the list of codes. A code was applied if it reflected a concept in a paragraph, i.e., microanalysis [38].
   d) Constant Memo-ing: Along step 3, notes were kept to capture key messages, relations, and observations on the study.

2) Selective Coding (2 Phases):
   a) Axial Coding: Comparing the concepts coded led us to inductively generate relations among coded concepts (e.g., “suboptimal organizational structure” causes “social debt” and so on.)
   b) Aliasing: The definitions of all concepts coded were compared with each other to identify aliases.

3) Theoretical Coding (Three Phases):
   a) Data Arrangement: Captured every portion of text that was coded with a code on a table.
   b) Data Modeling: The data synthesis was represented in diagrams. The diagram shows all the core concepts [i.e., code clusters resulting from axial coding, phase b)] and relations found.
   c) Theoretical Sampling: The diagrams and all the data at hand were analyzed and sorted, trying to identify recurrent patterns, underlying relations, and hidden meaning. Observation of the data was aided by the standard analysis methods, such as weighted frequency analysis (i.e., by analyzing the number of times certain concepts showed up against the number of interviews in which they were found) card sorting (by rearranging the hierarchy of types to let underlying relations show themselves), and conceptual modeling.

Coding was carried out by two independent coders to ensure intercoder reliability; the statistical content analysis and assessment index defined by Krippendorff [17] as $K_{\text{alpha}}$ coefficient was used to evaluate overlaps between the coding schemas. First, the method was applied by a junior research student that generated the pilot list of possible codes (39 codes). Second, a post-doctoral researcher recoded a fresh version of the entire data set using the pilot list of codes, resolving issues in concordance with the research student involved. The alignment index $K_{\text{alpha}}$ measured 0.84 well beyond the common threshold for reliability defined as 0.800.

2) Causality Modeling: To represent causality, the 6C model was applied [13]. The 6C model allows to represent empirical causality by relating six variables: Cause, meaning the event or circumstance that gives rise to a consequence; Consequence, meaning the effect produced by a certain cause; Condition, meaning the constellation of variables that need to be true in order, for a cause, to manifest into a consequence; Context, meaning the circumstances that form the setting of the causality function; Covariance, the set of conditions that produce a mutual variation with cause or effect; and Contingent, the event or artifact whose value is compromised by the consequences in the causality function. These six variables are to be found among core concepts.

For example, in Fig. 3, “Category” represents a community smell. The left-hand side represents the cause for “Category,” i.e., the set of circumstances that make “Category” evident and result into the consequences on the right-hand side. In addition to the causes, the smell “Category” might be subjected to a set of conditions (box on top of “Category,” see Fig. 3). Causes for “Category” exist in the context of the smell (on top of the conditions in Fig. 3). Let us assume that you might want to do something about the smell by influencing “Category” cause. Certain covariances (or coevolving factors) might occur (lower box under “Category” cause). Contingents (lower box on the right-hand side) represent the value or event influenced by the consequences of “Category” similar to covariances but for consequences.

From the total set of core concepts extracted during coding, existing causality dependences were identified for about 20% of concepts found. While, in most cases, the application of the 6C was straightforward, in some cases, “Covariance” and “Contingent” could not be established.

3) Interrater Reliability Assessment: Finally, coding was carried out by two independent coders to ensure IRR. First, the method was applied by a junior researcher that generated the pilot list of possible codes (39 codes). Second, a post-doctoral researcher recoded a fresh version of the entire data set using the pilot list of codes, resolving issues in concordance with researchers involved. The IRR assessment was constantly monitored for increase using the well-known Krippendorff’s alpha coefficient [18]—a final evaluation of the $K_{\text{Alpha}}$ indicated an IRR agreement of 0.83, higher than the standard reference score of $K_{\text{alpha}} > 0.800$.

4) Evaluation by Focus Groups: Finally, to evaluate the entirety of the results and contributions captured in this paper,
the notions and results in question were stressed within two industry–academy focus groups that assumed the connotation of Delphi studies [22], [31]. More in particular, the notions in this paper were presented in a lightning presentation of about 10 min to a demographically diverse group of industrials, sampled randomly and through self-sampling techniques—the sample is represented in Table I, which summarizes: 1) the role of the participant; 2) the number of experiences with social debt reported in his/her past; 3) the sector of his/her practice; and 4) the experience in years of practice. The conversation was steered by the focus-group facilitator over four main points, following the funnel-like convergence style, according to the guidelines of Runeson and Höst [44]. The four conversation points are reported in the following:

1) previous reported experience or incidents with respect to any notion under discussion;
2) software artifacts that were immediately influenced by any such notion;
3) types of people involved in social debt;
4) patterns and characteristics in the industrial experiences;

From the above-mentioned conversation points, the contents of all results were refined eliminating or rephrasing any notion that disconnected from consensus in the focus groups/Delphi study enacted for evaluation. Beyond this Delphi-study connotation, the evaluation group reported an interesting series of additional concepts stemming from the experiences of practitioners with architectural social debt. These additional contributions are reported in Section VII.

C. Triangulation Approaches in the Final Evaluation

The findings reported in this paper were attained interviewing and carrying action research over the development and software operations of a total of 13 individuals arranged in 4 roles, as specified in Table II. The exploratory nature of the research forced us to consider using observer triangulation as well as data-source triangulation. For observer triangulation, the data were observed by two people directly, and then, an IRR score was assessed featuring again the Krippendorff’s alpha coefficient [18]—the coding of the data reflected a $\kappa$-alpha score of 0.81, higher than the standard score of 0.800. Concerning data-source triangulation, the observations concerning each role were coming from multiple sources, for example, the community smells elicited were only considered recurrent if they were perceived by more than one role and more than one person covering for that role in the analyzed organizational structure.

IV. RQ1—ARCHITECTURAL COMMUNITY SMELLS

As previously mentioned, four recurrent series of circumstances were found, in which architecture decisions and the process of architecting reportedly generated social debt. Essentially, these circumstances represent the architecture community smells [7], [52], to be avoided if the social debt is not acceptable. This section reports on these smells, elaborating them with a simple box-and-line causal notation (see Figs. 4–7). These patterns are described using quotes and data (where possible).

A. Lonesome Architecting

This pattern was manifesting when nonarchitects are forced to make decisions, while actual architects are, quoting from one of the interviews, “too few and far apart.” One of the software architects reporting this condition also complained that he and his colleagues had not enough time to dedicate to decision-making (and related changes) as well as properly disseminating architecture decisions. This pattern is shown in Fig. 4. This pattern was found recurring over time in the project and was encountered five times. Some of the most common consequences found resulting from this pattern are: 1) decision unawareness; 2) misalignment between product version and architecture; 3) lack of awareness on the product’s needs; and 4) overly fast decision-making to “patch-up.” The debt, in this case, is associated with delays needed to find out about decisions and apply the necessary modifications, possibly rewriting code with considerable waste. Also, from a social point of view, this circumstance results in loss of project vision (i.e., frequent quotes were “what are we doing? what does the product need for its improvement?”) with resulting frustration and mistrust. For example, some developers reported that “sometimes [fellow developers] were...
Fig. 5. Obfuscated architecting, e.g., legacy systems means “legacy in mind” as well.

Fig. 6. Architecting by osmosis, semipermeable communication.

not even informed about some implementation made by the team of <GREEN> [after decisions were made] and only after things were done they received an outcome that needed to be analyzed.” This text suggests how architecture decisions were made without actually consulting with experts concerned with the decision. Also, some interviewees reported that “[the architects] decided for Java frames in <Integra>, but these created many compatibility problems, too many java versions available in the market, not always compatible [with the products’ current version]. Also, the frames really work according to the customer requests only with the newest [very expensive] state-of-the-art hardware.”

B. Obfuscated Architecting

Obfuscated architecting takes place when multiple subgroups (e.g., GREEN and RED in this case) emerge in a development network without a harmonized organizational and sociotechnical vision necessary to operate in the network.

This pattern was manifesting when new or changed architecture decisions imply implementation changes that necessitate new people to be included in the development network (e.g., different skills are needed). This pattern was observed in the presence of multiple products (both legacy and new) being operated together but in the process of being integrated. New people to be included in the development network lacked the frame of mind and vision needed to understand and cope with the legacy product. This obfuscated the communication of architecture decisions. This pattern is shown in Fig. 5 and was found occurring 14 times.

Some of the most common consequences found resulting from this pattern are: 1) single communication points for architecture decisions—many developers eventually felt left out of the development network when it came to software architecture, and since they could not reach architects properly, this led to time waste and resulting developers’ frustration and 2) circumstances indicating sociotechnical code churn, much similar to reports by Meneely and Williams [29].

For example, some developers reported that “[certain developers] do not know who the architects are, apart from [a certain architect] and how to provide feedback to them, following which process. There is maybe a process, documentations are there, and some of them are written very well, but they are not properly used.” Also, integration engineers from the interviews reported that “[they] get the feeling that no one is overseeing the situation. For the first time, now [after a long time], we are going to have the first meeting with people who are preparing specifications for <RED+GREEN Integration> to present to them how they have to run [some component].” Finally, some architects reported that “some implementations are done also considering the future implementation regarding the data layer. [Other teams] are responsible for new requirements coming from a customer […] for <GREEN>. So only partial design solutions were made [in a certain version], while the migration decision (still happening) was taken from the product management, even if it was decided before the [newer version of] integration came out!”

C. Architecting by Osmosis

In layman’s terms, osmosis refers to the process of permeating a solvent through a semipermeable (series of) membrane(s). By comparison, architecting by osmosis means making architecture decisions using the knowledge that is filtered through many semipermeable communication links.

Architecting by osmosis was observed manifesting when the following sequence of events occurs: 1) the effects of certain decisions reach clients and product operators but result in inoperable software; 2) operators, pushed by clients, share malcontent with developers and suggest technical changes; 3) developers evaluate (and sometimes partially implement) possible technical changes and suggest change to architecture decisions; and 4) architects make necessary changes in decisions with knowledge that was partially filtered by all communication layers in the development network. This pattern is shown in Fig. 6. It was found recurring over time.

---

and was encountered three times. Some of the most common consequences found resulting from this pattern are as follows.

1) **Decision Localization:** Architecture changes were made by certain architects who did their best to communicate, but the decision (and consequent refactoring) remained local to their own team—apparently architects assumed that the same communication chain for the change request would disseminate the change itself—eventually, this phenomenon resulted in uncooperative behavior.

2) **Poor Decision Documentation:** Much of the rationale and reasoning behind architecture change requests went lost as people filtered out details through every communication link, and this reportedly caused frustration when architecture documents were involved and eventually led to deprecation of architecture documents.

3) **Architecture Erosion:** Some architecture decisions were changed as they created unstable configurations, and this reportedly caused a great deal of misalignment.

   For example, some developers reported that “[many times] decisions are only commented in person with people across the corridor, informally and they spread the word into their own competence team.” Also, some architects reported that colleagues were “[about documenting architecture changes and requests] probably using different processes and standards according to software change request documents that [they were] receiving [describing changes to be made], showing a different and confusing syntax.” Finally, some interviewees reported that “[some component] is already in production for over 10 years now and there are still people, domain specialists [and architects] who are not aware of that and still making specifications based on the old [some component model], and they are so used to that and they do not think about anything else.”

**D. Invisible Architecting**

Invisible architecting takes place when architecture knowledge conveyors used by development newcomers become oppressive rather than clarifying. This circumstance comes about when architecture documents or architecture knowledge conveyors (e.g., minutes of meeting) are not used consistently either by the current development network or by team additions themselves. The result is that the process of architecting and the knowledge necessary to enact this process, e.g., reasoning, rationale, and assumptions, although present and explicit somewhere, become invisible.

This pattern recurred five times across the data set. In one instance, for example, a developer added to the development network documented the implementation of a decision in loosely formatted text rather than using specific forms. In another instance, for example, an architect misinterpreted his own decision taken years past only later remembering that rationale was to be found in the implementation. The resulting social and technical debt effects are: 1) decision unawareness—developers were not fully aware of architecture decisions; 2) product versus architecture misalignment—some developers found themselves working on wrong or unsupported versions of architecture components or related components-off-the-shelf; and 3) solution defiance—software developers or operators refused to accept architecture decisions, a circumstance previously reported in [52] and leading to social debt.

For example, some developers reported that “different teams are not communicating [with the architect or among themselves] even if they are supposed to work in parallel and use [some versioning tool]. They try to solve their own process and they do not care about [some architecture decision for which] they are not responsible for. You can see lack of vision, no collaboration.” This pattern is shown in Fig. 7. Similar to obfuscated architecting (see Section IV-B), invisible architecting shows some peculiarities: 1) it relates to the usage of architecture knowledge conveyors rather than cognitive distance in mindset; 2) the observable effects were reported as causing more delay; and 3) architecture incommunicability manifested for people or teams added to the development network, while invisible architecting was rapidly disseminating across the entire development network.

Fig. 8 shows the occurrence of above-mentioned patterns to evaluate their different context. First, when the architecting layer is relatively silent and distant from the rest of the development network, *lonesome architecting occurs*. Second, when architecture change requests come across every possible
communication layer in the network, *architecting by osmosis occurs*. Third, when multiple subgroups are present in the development and operations layer, *obfuscated architecting occurs*. Finally, when knowledge conveyors are not sound, *invisible architecting occurs*.

**Summary for RQ1:** There exist at least four circumstances in which social debt manifests in a manner which is connected to software architectures, namely: 1) lone-some architecting; 2) architecting by osmosis; 3) invisible architecting; and (4) obfuscated architecting. The overarching characteristic that underlies all architecture community smells is the general inability to communicate software architecture decisions.

V. RQ2—INCEPTING INCOMMUNICABILITY

In the process of addressing RQ1, the act of sharing architecture knowledge was observed as the process of making available software architecture decisions and related artifacts [15]; the aforementioned process is seen by all as a necessary quality of software processes and resulting products but itself is not sufficient to warrant for reduced social debt.

**Architecture Incommunicability**—$A_{Inc}$: The inability to communicate architecture decisions directly to those who should be aware of them—the inability is due to adverse organizational or social circumstances across the development network.

In other words, $A_{Inc}$ is the it is a % of organization that incurs the risk of not receiving a direct communication of the architectural decision, that is, the number of people who are working or somehow directly connected to components whereby a decision is being taken $D$, which do not have strong ties to the decision-maker $T$, averaged per every decision $D$ within a certain time frame $T$.

By its very nature, this notion of *incommunicability* is related to communication and, hence, is afflicted by social and organizational circumstances (e.g., organizational filtering protocols or nondisclosure agreements).

A. Roots of Architecture Incommunicability

As previously stated, architecture incommunicability is a measurement of the degree to which architecture component workers are strongly tied with architecture decision-makers acting over those components. As such, *architecture incommunicability* can be studied combining principles and techniques from the SNA with an analysis of who makes architecture decisions, as opposed to who actually knows about said decisions. The scenario seems consistent with what is known as "weak-ties hypothesis" as elaborated by Granovetter [14] in SNA (see Fig. 5). Paraphrasing from [14], the weak-ties hypothesis implies that the following holds.

**Weak-Ties Hypothesis:** "If entity A is linked to both B and C, then there is a considerable probability that B and C are related as well, with a weaker but identical relation."

As a parallel for architecture-level incommunicability, and in the scope of the case study reported in this paper, only the people who are *strongly tied* with the decision-maker were made aware *directly* of the decision, that is, in a
short time span. Conversely, people working on components depending on those upon which a decision was being taken (see Fig. 11, Dev. 2 working on C4 which is depending on C1), exhibited weak ties with decision-makers or their strongly related coworkers; these individuals found out about architecture decisions only eventually after a longer time span. For example, stemming from Fig. 9, the decision-maker A is connected to two ancillary roles B and C, which, in turn, are connected to their own ancillary roles across the development network; the weak-ties hypothesis assumes that while A−→B and A−→C are direct connections, there exist also other connections: 1) A is also weakly tied to B and C’s ancillary roles who, however, receive information at a second stage and 2) B and C are also weakly tied with each other, since they both receive information concerning an architecture decision that influences their respective components and, hence, they have implicit or explicit social, organizational, and technical dependences. The time distance between receiving software architecture communications through strong ties or weak ties, respectively, constitutes the hypothesized architecture incommunicability.

In essence, the strong ties intended here constitute the social or organizational ties among decision-makers and developers, while the weak ties exist between decision-makers and developers through social, organizational, or technical dependences (e.g., across components affected by the decision, APIs, and so on), which are not backed up with explicit social or organizational ties.

Generalizing the above-mentioned observation and phrasing it as a general hypothesis under which all architecture decisions are being taken, a preliminary assessment over \( A_{inc} \) can be made. More specifically, analysts can observe and compute the difference between: 1) the number \( N \) of people who are, on average, strongly tied to the decision-maker for any set of decision D* and 2) the average number of people who should directly be aware of decisions D*. This difference procures an estimate of architecture incommunicability.

\[ \text{METRIC 1:} \quad \text{Given a certain decision D, understand which components are related to it and infer who is currently working or responsible for said components. This measurement is a percentage of people in the development network, i.e., a decision popularity metric ( DP).} \]

\[ \text{METRIC 2:} \quad \text{Given a certain decision D, understand the subgraph of the development network that is aware of a certain decision assuming the weak-ties hypothesis [14] for developers related to the decision-maker. This is again a percentage of people in the development network, i.e., a decision awareness metric ( DA).} \]

\[ \text{METRIC 3:} \quad \text{Given a DP average for all decisions in a certain time frame and its related DA average, compute the gap between the two—this amount indicates the percentage of people across the development network that should know but are almost certainly unaware of decision D, i.e., the mean architecture incommunicability ( AI) indicates the mean percentage of people that should be aware but, in fact, are not. In this vein, this quantity indicates the actual social debt across the network since: 1) it represents the fixed amount of additional cost incurred regularly (e.g., monthly, per every decision, per every release, and so on) across the organizational structure; 2) it is reduced to zero (i.e., paid off in total) when the organizational structure is refactored to accommodate for the reported social debt (e.g., the connected community smells have been addressed); and 3) it increases the fixed amount, or principal, connected to every social debt item (e.g., a single community smell).} \]

\[ \text{METRIC 4:} \quad \text{Compute the Average Per-Person Delay (APeND) connected to each instance of the reported community smells or connected to decision unawareness smells in the case of the antipatterns reported in Section IV; this is a measure of an additional cost over time. The cost in question applies to a people dyad, since both people involved in an instance of a community smell suffer from its effects. Also, the quantity can be computed multiplying man-hours (the actual delay connected to the community smell) times the full-time equivalent (FTE) cost for that time. Once the number is available, a multiplication of APeND (mean cost) and AI (mean number of unaware developers) amounts to a measure of the architecture social debt connected to incommunicability, in the case of the community smells from Section IV.} \]

More formally, DAHLIA architecture incommunicability can be expressed as follows:

\[ AI = \|DP - DA\| \quad (1) \]

that is, the absolute difference between decisions’ popularity and decisions’ awareness. Similarly, a measurement of the
connected social debt can be obtained in the following form:

\[ A_{S\text{Debt}} = A_I \times \text{APPeND} \times T. \]  

(2)

From a software engineering economics [6] perspective, (3) is remarkably similar to the basic equation for loan balance contraction [37], which is outlined in the following format:

\[ i = P \times r \times t \]

where the total interest “i” is calculated as the Principal (the fixed amount which is loaned) multiplied by the interest rate “r”, that is, the rate or spread of the interest across a timeline of reimbursement “t.” Within DAHLIA, the following mappings apply: 1) rate “r” is the spread of the debt across the development network or \( A_I \); 2) a consistent part of the interest “r” is also the quantity connected to the community smell or another social debt item in question, represented by the APPeND measure; and 3) the time \( T \) is the number of the smelly decisions taken over the duration of project lifeline until the point of analysis “t” considered by the analyst.

Conversely, from a social-networks’ perspective, the reasoning behind DAHLIA is conceptually identical to the logic behind transactive-memory systems (TMSs) previously defined in social networks and organizations research by Wegner [58]. Essentially, TMSs are mental mechanisms through which groups collectively encode (by capturing actors and actions, “who did what”), store (by capturing the experts, “who should know what”), and retrieve knowledge (by retrieving the experts, “who knows what where”) [59], contributing to the creation of a group mind or collective intelligence [59].

In layman’s terms, DAHLIA is a framework to measure architecture communicability and the connected social debt using a continuously evolving TMS of architecture decisions (see Table III).

In summary, the relations around social debt and its estimation are worthy of attention and several remarkable similarities exist with the theories of software engineering economics as well as the most advanced theories around architectural technical debt (ATD) [25], [26]. For example, as previously stated, Martini and Bosch [25] have already offered a reinterpretation of fundamental debt economics theories to further understand and map to technical debt. This paper and the DAHLIA framework may serve as a rudimentary basis in the same research direction.

C. Applying and Automating DAHLIA

From an automation perspective, the TMS formed by DAHLIA can be obtained combining two sociotechnical artifacts related to software architecture and computable during software lifecycles as follows.

1) A structural dependence view of the software architecture components augmented with: a mapping of architecture decisions onto related components and a mapping of how many developers/operators are currently working or responsible for said components. This view relates architecture components among themselves (using their associations or dependences) with decisions concerning said components. A sample of this view computed using data from the case study can be seen in Fig. 10. This view is needed to measure how many people should know about a certain architecture decision (and the connected DP), as defined earlier—also, this view can be easily generated by augmentation of structural dependence views automatically generated by tools, such as TITAN [61] or Understand.\(^7\)

2) A social-network representation of the development network computed using strong interaction and collaboration relations existing among developers. This representation for the scenario at hand was previously introduced in Fig. 1. This view is needed to measure how many people actually know about a certain architecture decision (and the connected DA), assuming the existence of strong ties between the decision-maker and other members of the development network—also, this view can be easily generated by tools, such as CodeFace [16] or Butterflyzer.\(^8\)

Applying DAHLIA means following the process next.

1) Evaluate and subtract \( D_P \) and \( D_A \) for all architecture decisions taken up to the instant under investigation.

2) Compute an average of the above subtractions, i.e., mean architecture incommunicability (\( A_I \)).

\(^7\)https://scitools.com/feature/metrics/

\(^8\)http://butterflyzer.com/
3) Compute the APeND value for the project at hand.
4) Multiply APeND and the number of people reflected by $A_I$, i.e., additional project cost connected to social debt.

Note that the above-mentioned process can be applied even with an *a posteriori* evaluation. In this case, observers would need to compute and subtract $D_P$ and $D_A$ for every architecture decision made in the observed project.

To illustrate DAHLIA, Section V-D applies the framework in practice using data and models from the case study.

### D. DAHLIA in Action: A Scenario

Using case-study data from DAHLIA could be applied in practice for a limited number of decisions (three decisions) both before and after the usage of Architecture Boards (ABs). This allows us to: 1) illustrate DAHLIA and its workings on a limited yet expressive scenario and 2) illustrate a rough estimate of the effect that using ABs introduced in the scenario at hand.

A tableau with the complete working of the DAHLIA application in this case study is available online to encourage replication and verifiability.\(^{10}\)

First, let us assume that D1 is a decision associated with component C1. Also, let us assume that component C1 is related to components C3–C5. This scenario is represented in Fig. 11. *What is the incommunicability for D1?*

According to the definitions from Section V-B and the assumptions mentioned earlier, there are at least eight people who should be made aware of D1 as follows:

1) one developer who is currently working on the same architecture component on which the decision is acting;
2) seven developers working on components related to the one affected by D1.

Eight out of thirteen people mean that 61% of the development network should be aware of D1. This is DAHLIA $D_P$ metric. However, by virtue of strong ties, people who are strongly tied with “Arch. 3” (see Fig. 11) are extremely likely to be be made aware directly about D1. According to the organizational–social network in Fig. 1, there are five such people strongly tied to “Arch. 3,” and hence, about 38% of the development network (this percentage is DAHLIA $D_A$) is more likely to be made aware of the decision in a most immediate time span.

By virtue of the weak-ties hypothesis, subtracting the two numbers gives a rough estimate of how many people should know but actually do not in a short time span, i.e., the $A_I$ value sought for: $8 - 5 = 3$, i.e., around 23% of the network. These people are more likely to know of D1 eventually after a longer time span. This exercise was iterated on the data for Integra for available decisions and found that, on average, two people part of the project were not aware of architecture decisions as they were taken or changed. Now, let us assume that the patterns causing decision unawareness yield an average delay of four man-hours (DAHLIA APeND value), as observed in previous work [52]. This delay is connected to social debt across Integra that amounts up to 150 Euro which is 0.5 FTE cost for a single day of a developer in the context of the Integra project. What results after the necessary calculations is an additional cost of 300 Euro per every smelly decision, connected to an $A_I$ average value of 20%.

### E. DAHLIA Strengths and Limitations

First of all, DAHLIA may potentially be automated combining tools for sociotechnical analysis of development networks with SNA techniques. Many if not all software processes nowadays already have all the data (sometimes even publicly available in the case of open source) to apply DAHLIA.

---

**TABLE III**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Name</th>
<th>Definition</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_P$</td>
<td>Decision Popularity Metric</td>
<td>Given a certain decision $D$, understand which components are related to it and infer who is currently working or responsible for said components.</td>
<td>Development Network Ratio (%)</td>
</tr>
<tr>
<td>$D_A$</td>
<td>Decision Awareness Metric</td>
<td>The subgraph of the development network that is aware about a certain decision assuming the strong ties hypothesis for developers related to the decision-maker.</td>
<td>Development Network Ratio (%)</td>
</tr>
<tr>
<td>$A_I$</td>
<td>Mean Architecture Incommunicability Interest</td>
<td>Given a $D_P$ average for all decisions in a certain timeframe $T$ and its related $D_A$ average, compute the gap between them - this indicates the percentage of the developer network affected by the debt, hence, the <em>interest</em> connected to the debt, increasing per every decision taken without any change in the org. structure.</td>
<td>Absolute Ratio (%)</td>
</tr>
<tr>
<td>AP-PEND</td>
<td>Average Per-Person Delay</td>
<td>Measure of an additional cost over time, and can be measured in man-hours times the Full-Time Equivalent (FTE) cost for that time.</td>
<td>Man-Hours Cost ($)</td>
</tr>
<tr>
<td>$A_{Debt}$</td>
<td>Mean Architectural Incommunicability Social Debt</td>
<td>A measure of the additional project cost accumulated due to architecture incommunicability and connected to all decisions incommunicability until time $T$ of the project lifecycle.</td>
<td>Debt Interest Ratio until time $T$ ($)</td>
</tr>
</tbody>
</table>

---

9 All data and models were properly anonymized, reshuffled, and modified according to the nondisclosure agreements.

10 https://tinyurl.com/dhalia-application
such as task-to-component allocations, or organizational–social structure, e.g., by combining the approach in [30] with social-networks’ mining techniques [28]. Further research into DAHLIA might reveal further insights into the relations between social debt and software architectures.

Although showing promise, DAHLIA has a number of intrinsic limitations. First, as previously observed, it can become difficult to pinpoint who is making decisions and what decisions are made about software architectures. Given that for reliable measurement, DAHLIA should be applied to every architecture decision and over time. This might become difficult to be applied in practice. In addition, the assumptions made about who should know certain architecture decisions might be overly optimistic and merely reflects the bare minimum of people that should be made aware about certain decisions. In fact, knowing who should know about which architecture decisions is based on assumptions that are forced by the context of work and the development problem at hand. These assumptions might not be clear until well into the software development process. Moreover, more research is needed to establish and validate the assumptions and logic through which DAHLIA was constructed, since the same SNA assumptions that were used might not be usable in other scenarios or contexts. Similarly, some parts of the reasoning behind DAHLIA are intrinsically limited in their ability to predict cost. For example, APPeND values are means associated with delays. These delays are not necessarily related to social debt and definitely not necessarily related to architecture decisions and their communicability. More systematic research is needed into confirming the reasoning behind DAHLIA and its validity before its use may be generalizable.

**Summary for RQ2:** The reported community smells cause an amount of social debt connected to the application of the metrics framework called DAHLIA; the DAHLIA framework can be used to manage the debt connected to the reported architectural community smells.

VI. RQ3—Mitigating Architectural Social Debt

RQ3 sought to determine which mechanisms did the architects and practitioners in the Integra case put in place to address their perceived architectural social debt. We observed that a large part of the negative and invisible effects reported in Section IV were mitigated with success in Capita using a practice referred by people part of Integra as an AB. This section elaborates more on ABs and discusses/evaluates their uses and limitations by means of DAHLIA.

**A. On Architecture Boards and Their Efficacy**

An AB is essentially a subcommunity in Integra comprised of people who were, are, or are likely to be responsible for architecture decisions and their dissemination. More in particular, any person involved in the project and matching the following prerequisites would become AB member.

1. The person has made or influenced architecture decisions with provable rationale.
2. The person has architecture expertise or concerns and belongs to a site without an AB member.
3. The person has architecture expertise or concerns and belongs to a team without an AB member.

Also, for certain architecture decisions requiring particular domain-specific expertise, client or domain-specific analysts’ intervention was required for one or more AB meetings.

Thus structured, an architecture board is consistent with the Problem-Solving Community previously reported in [55]. The community met on a biweekly basis to fulfill two organizational goals: 1) make and disseminate architecture decisions and 2) gather and monitor the usage of organizational culture inherent to devising and disseminating decisions.

This practice reportedly reduced debt connected to the three patterns reported earlier, namely: 1) obfuscated architecting; 2) invisible architecting; and 3) lonesome architecting.

**B. DAHLIA Analysis**

To evaluate the usefulness of DAHLIA as a mechanism to assess and manage social debt, the mechanism was used to evaluate the effect of the same decisions in the presence of an AB, which acts as a community for architecture knowledge dissemination and maintenance of software-architecture-related organizational culture.
Considering the example for D1, besides the five people strongly tied to “Arch. 3,” the decision-maker (see the organizational–social network in Fig. 1), an additional set of two people are made aware of the decision in the architecture board.11 It follows that a total of seven people are made decision-aware against the eight people who should be made aware of architecture decision D1.

Again, by virtue of the weak-ties hypothesis, subtracting the two numbers yields an estimate of how many people should know but actually do not in a short time span: $8 - 7 = 1$, i.e., around $8\%$ architecture incommunicability across the network, a considerable improvement of $15\%$ additional architecture communicability.

### C. Negative Consequences of Architecture Boards

Nevertheless, using ABs was not without nasty consequences. For example, some board members eventually assumed subversive behavior of their own, e.g., pulling decisions and decision-making toward their own concerns while disregarding or belittling others, with consequent emergence of social debt. Quoting from the interviews that “[members of the board] are essentially different architects from different teams and pulling toward their own direction instead of finding a common standard [for organizational structure and goals],” Reports of this circumstance were found four times. Also, the AB itself received little or no formal recognition by the organizational structure around Capita. This reportedly compromised its existence. Indication of this was found five times in the interview data. For example, one architect reported that “the architects board is also unofficial. [this means] no formal organization, no discipline. [Hence], still there are cases in which architects are not involved and requirements are not well documented and specified. […] Involvement of architects always! To minimize the risk of false implementation.”

### Summary for RQ3:

Architecture boards are the specific groups of software architects and other interested parties; the group in question can be used as a mitigation mechanism for the nasty effects connected to the community smells reported for RQ1, but its application is not bullet proof and does demand a tradeoff analysis and organizational evaluation.

---

11This number can be calculated evaluating how many people from Fig. 1 are part of the AB.
requires more and more organizations to adapt themselves in continuity with their software counterparts—little is understood of this interplay. Social debt, incommunicability, and SNA of community smells could all be put to practical impact and use for the purpose of better steering and managing microscale to macroscale software architectures behind modern IT.

3) Theoretical Limitations: Although DAHLIA does, in fact, produce benefits in estimating and allowing the management of (some of) the social debt in software-intensive organizations, the theoretical underpinnings behind the framework show several limitations.

First, DAHLIA is only useful concerning the social debt items connected to incommunicability, for example, the community smells that introduce communication barriers (e.g., Organizational Silo Effects [51], [52]) or lower communication among software practitioners (e.g., the circumstances highlighted in Section IV). This is notwithstanding that social debt is a complex phenomenon with many ramifications toward technical debt—the above-mentioned quantities only go as far as allowing management of some of the debt but do not say anything about how much it is in total or how to elicit all of it for further management. This is a serious theoretical limitation behind DAHLIA to be tackled with future work.

Second, the weak-ties hypothesis has been shown to reflect complex social and organizational phenomena and is, therefore, a valuable asset to build upon when understanding social debt. However, the same hypothesis was never actually proven, in general, in social networks theories. Further research into social debt should take this limitation and its ramifications into account to properly counterbalance the theoretical modeling around the social debt phenomenon.

Third, although it is theoretically possible to automatically measure social debt connected to incommunicability as introduced in this paper, gathering and processing the information required are only worth pursuing within large and complex organizations where the benefits of addressing social debt are also higher. In general, frameworks, such as DAHLIA, are not readily applicable to small–medium enterprises where social debt itself may even become a positive phenomenon much like some technical debt is an acceptable risk in the same circumstances. Further research into this limitation and its meaning with respect to social debt is required.

4) Threats to Validity: Based on the taxonomy in [60], four categories were found as potential validity threat areas, namely, external, construct, internal, and conclusion validity.

a) External validity: It concerns the applicability of the results in a more general context. Being this study performed in one organization, and using qualitative means alone featuring interviews that are subjective by nature, results could be specific to the particular context of investigations. To reapply results and possibly confirm the validity of this study, several reapplications and replications are being planned, with additional independent explorative case studies featuring quantitative research as well. Also, DAHLIA and connected results were only investigated in the scope of the scenario at hand and must be validated more thoroughly before the framework’s validity is ascertained.

b) Construct validity and internal validity: They concern the generalizability of the constructs under study, as well as the methods used to study and analyze data (e.g., the types of bias involved). To mitigate these threats, the methods were tailored to use multiple triangulations of data sources, intermixing interviews with focus groups and confirming observations and partial results through subsequent workshops. Also, a representative from the management of “Integra” verified the interpretations of the data and provided clarifications and corrections where necessary. In addition, partial results and incremental analysis were conducted to gather constant independent feedback by three senior researchers. Finally, since DAHLIA and connected metrics and observations were defined through basic means with little validation, both their internal and construct validity might be at stake. Several additional studies are being planned to confirm the validity of the proposed framework and its link to measuring social debt. Also, the materials, in this paper, are currently in the process of demonstrating the representation condition [12] for metrics contained in the framework.

c) Conclusion validity: It concerns the degree to which the conclusions are reasonable based on the data. The conclusions were drawn by an analysis of empirical evidence using the known and confirmed methods from the literature, such as coding, gap analysis, and taxonomy analysis. Nevertheless, being the results and conclusions drawn from an interpretation of qualitative data alone, this might compromise validity. In addition, DAHLIA and connected metrics were developed and tested out in the scope of the same case study, and hence, the validity of conclusions connected to this result should take the context of investigation under careful advisement.

VIII. PREVIOUS AND RELATED WORK

The preliminary elaboration of social debt stems from a series of intuitions and case studies reported in [51] and [52], respectively. Tamburri et al. [51] carried out scenario and factor analyses to understand whether technical debt might have a social counterpart and how were social and technical debt related. The authors concluded that we barely scratched the surface of a mountain, to explore which needed more systematic empirical study. Tamburri et al. [52] made a first attempt at understanding social debt emerging in industrial practice, defining a framework to interpret and further investigate the concept. The results in this paper are consequent to digging deeper in the previously unanalyzed data concerning software architectures, architecting and architecture decisions that are originated from the same data set of [52]. The results obtained from the new analysis in this paper are completely novel and reveal ways in which not only software processes but also product architectures and architects themselves are linked to the social debt phenomenon.

As previously stated, from an organizational and sociotechnical perspective, the contact points and implications between organizational–social structure and software success (in terms of products, processes, and people) are seeing increased attention over the last few years. For example, Nagappan et al. [33]
show, in practice, the influence of organizational structure and other “human” aspects on software quality. This and similar works (see [41] or [57]) bring evidence that motivates the study of social communities in organizations and the debt (if any) connected to them. This family of studies contributes to the social debt by providing evidence of its existence and impact. In addition, these studies provide the valuable data to identify the orders of magnitude that regulate social debt. The study in this paper is related to the results noted by Nagappan et al. [33], which were observed in a live organizational and social structure, but chose to focus on the software architecture it reflected. While Nagappan et al. [33] limit to assess the impact of organizational structure on software quality, the study captured in this paper strived to focus on the patterns of suboptimality across said architectures, e.g., as a consequence of low-performing team patterns [21] or suboptimal organizational structures emerging in a global workforce [56].

In addition, some works are emerging that discuss software architectures and how they can be used to drive processes, people, and products to an alignment which avoids social and technical “friction,” e.g., the work by Nord et al. [36]. Here, the authors discuss how software architecture can be used to instill agility in large-scale software development endeavors. In this case, software architecture is to be studied from three interrelated perspectives, namely: 1) system; 2) development organization; and 3) production infrastructure. Stemming from this premise, the work discusses the ways in which software architectures and the process of architecting might lead or be subjected to undesirable or unwanted patterns appearing at the sociotechnical and organizational level. Following a similar inquiry line, Martini and Bosch [26] have been active for quite some time in the investigation of ATD and the connected organizational phenomena. For example, Martini uncovered the notion of vicious circles and contagious ATD. As an outcome of this and similar studies many have attempted at elaborating metrics for the management of ATD [25], [35].

In a similar research direction to find predictive and assessive mechanisms for ATD, Liang and Yang [20] and MacCormack and Sturtevant [23] have elaborated metrics frameworks to assess the cost and impact of ATD by means of quantitative empirical studies. By way of comparison with these studies, this manuscript proposes a way in which software architectures, architecture decisions, and the process of architecting can be instrumented for SD prediction. Finally, the manuscript introduces and discusses a mitigation mechanism and a metrics framework, DAHLIA, to help manage and reduce some of the nasty consequences of the architecture community smells illustrated in Section IV.

**IX. Conclusion**

This paper reports on patterns in the process of architecting and ways in which the connected debt can be measured and managed. The patterns in question were observed in an industrial case study and found that they result in what is called social debt, i.e., a picture of the current state of things burdened by suboptimal sociotechnical decisions. In the case of the patterns reported in this paper, a considerable part of architectural decisions or the very process of architecting in a certain way had a negative and (previously) invisible sociotechnical connotation. This paper also reports on a practice observed in the case study to mitigate some of the nasty consequences connected to emerging social debt. Finally, the manuscript outlined DAHLIA, a sample metric for architectures to make (some) social debt explicit by measuring architectural (in)communicability, i.e., the likelihood that the developers’ network is (un)aware of architecture decisions.

This paper contributions and discussions lead to the conclusion that social debt and software architectures are tightly knit together and with technical debt as well and deserve further attention in the future. Further studies are planned to elaborate on the findings, expanding the understanding of the relations between social debt, its technical counterpart, and software architectures possibly with an immersive study in industry. Also, further studies are planned to understand the differences between social debt in closed-source industries (as reported in this paper) and open source, to possibly find successful architecting patterns elicited from open source, if any. In addition, several points highlighted in this paper remain open, for example: 1) what is the full mapping between software engineering economics concepts and the notion of social debt? 2) what is the role of interest repayment strategies as well as economic formulations around debt contraction and repayment as mapped to social debt? 3) what is the relation between effort-estimation techniques, the consequent organizational structure decisions, and the emergence of social debt?

Finally, further studies are being planned to elaborate further on the proposed metric for software architecture communicability, demonstrating its representation condition [12] and applying it in industrial practice in the operational form, with the goal of providing more solid and quantitative validation for it.

**Acknowledgment**

The author would like to thank Dr. F. Castri for his invaluable contribution to obtaining and confirming the results presented in this paper. He would also like to commend the help of senior researchers that participated in the design and execution of this study, mainly Prof. Dr. E. Di Nitto, as well as senior practitioners from CAST Software, namely, Dr. M. Crubellati, Dr. M. Bessi, and Dr. B. Curtis.

**REFERENCES**


**Damian A. Tamburri** received the Ph.D. degree *cum laude and mention* from Vrije Universiteit Amsterdam, Amsterdam, The Netherlands, in 2014, one year in advance of a standard Ph.D. Contract. He is currently an Assistant Professor with the Jheronimus Academy of Data Science, ’s-Hertogenbosch, The Netherlands. Though still in his very early career, he has published over 100 papers in either journals, such as the *Transactions on Software Engineering Journal*, The *ACM Computing Surveys* Journal, the *IEEE Software Magazine* or top software engineering conferences, such as ICSE or FSE, and top software architecture conferences, such as ECSA or WICSA. He has also been an active contributor and lead research in many EU FP6, FP7, and H2020 projects, such as S-Cube, MODAClouds, SeaClouds, DICE, ANITA, DossierCLOUD, and ProTECT. His current research interests include complex software architectures (with a focus on data-intensive architectures, cloud and microservices, and machine learning and computational intelligence architectures), complex software architecture properties (with a focus on privacy and security), and empirical software engineering (with a focus on organizational, social, and societal aspects with qualitative methods, social networks analysis, and machine learning).

Dr. Tamburri is an IEEE Software Editorial Board Member, a Secretary of the TOSCA TC, and the Secretary of the IFIP TC2, TC6, and TC8 WG on Service-Oriented Computing.