Tamburri *et al. Journal of Internet Services and Applications*
DOI 10.1186/s13174-015-0024-6

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH Open Access

Social debt in software engineering: insights from industry

Damian A Tamburri[1*], Philippe Kruchten[2], Patricia Lago[1] and Hans van Vliet[1]

Abstract

Social debt is analogous to technical debt in many ways: it represents the state of software development organisations as the result of "accumulated" decisions. In the case of social debt, decisions are about people and their interactions. Our objective was to study the causality around social debt in practice. In so doing, we conducted exploratory qualitative research in a large software company. We found many forces together causing social debt; we represented them in a framework, and captured anti-patterns that led to the debt in the first place. Finally, we elicited best practices that technicians adopted to pay back some of the accumulated debt. We learned that social debt is strongly correlated with technical debt and both forces should be reckoned with together during the software process.

Keywords: Software project management; Social debt; Socio-technical decisions; Social software engineering; Case study

1 Introduction

Software engineering success is increasingly dependent on the well-being of development communities [1]. In some of our previous work [2-4], we found many decisions influencing community well-being. For example, changing the organisational structure [5] of the development community (e.g., through outsourcing), changing the development process (e.g., by adopting agile methods), leveraging on global collaboration (e.g., by striking a balance between formal and informal communication across global sites) are all socio-technical decisions, i.e. social and technical at the same time, that influence the state and welfare of developing communities and their members [5]. The social connotation of these decisions, changes the way people work and interact with others - i.e., their organisational and social structure [4]. The technical connotation of these decisions, changes the way in which development tasks are worked out. In agile methods, for example using Kanban boards, a "pull" task-allocation is often used, as opposed to classic "push".

Some socio-technical decisions eventually cause additional cost (e.g., through delays) on software projects and the development community around them. This cost, is not immediately visible and its resolution is often postponed. Also, if the same decisions remain into place, project costs may increase. For example, organisations often employ strict information-filtering protocols to protect industrial secrets when they embark in open-source [6]. This decision, however, might slow down development interactions, causing delays or even inciting fear.

This extra cost is conceptually similar to technical debt [7], i.e., the additional project cost caused by sub-optimal technical decisions. However this extra cost is not necessarily related to code and it is actually "social" in nature, i.e. connected to people and their development organisation. Paraphrasing Cunningham [8] who first introduced technical debt, social debt can be thought as: "not quite right development community - which we postpone making right". In Layman's terms, by social debt, we indicate the additional cost occurring when strained social and organisational interactions get in the way of smooth software development and operation.

While technical debt has received increased attention over the last 10 years, this other form of debt, namely "social debt", has remained latent and relatively unexplored. For instance, software engineering practitioners still lack a way to formalise socio-technical decisions and measure the connected debt.

This article takes one small step forward towards defining and studying social debt by means of an industrial

*Correspondence: damianandrew.tamburri@polimi.it
[1]VU University Amsterdam, Amsterdam, The Netherlands
Full list of author information is available at the end of the article

exploratory case study. The case study was driven by the following research questions: "What are the factors at play around social debt during the software lifecycle? Are there patterns in said factors? Can they be mitigated?"

From our study we learned that, indeed, there is a strong correlation between social debt and sub-optimal characteristics in organisational-social structures behind software development communities. Also, social debt is inextricably related to technical debt in many ways, e.g., uninformed socio-technical decisions generate both social and technical debt in a compounding manner that cannot be trivially "payed back". In addition, our findings suggest the presence of community "smells", i.e., precursors to the emergence of social debt, much like code smells may lead to technical debt, e.g. as shown in [9] by Zazworka et al. Finally, our findings uncovered some "mitigation strategies", i.e., ways in which some community "smells" may be averted and avoid the debt, even if partially.

In summary, this article offers four contributions: (1) a framework to define and interpret social debt; (2) a list of community "smells", intended as socio-technical anti-patterns that may appear normal but in fact reflect unlikeable community characteristics, such as anti-social organisational behaviour across the community, e.g., developers that refuse or delay information sharing; (3) a list of mitigations to said community "smells", as emerging from our case study; (4) a list of lessons learned as part of our case study that further discuss social debt and its implications.

The rest of the article is structured as follows: Section 2 outlines previous work related to the study of social debt. Section 3 outlines our study design, discussing our research methods and materials. Section 4 presents our contribution while Section 5 discusses their implications, pointing out to relevant threats to validity and ways in which they were tackled. Finally, Section 6 concludes the article.

2 State of the art

On of the very first attempts at linking software and the social/organisational processes around it is represented by Conway's Law [10]. According to Conway's law, software mimics (and sometimes is almost isomorphic to) the organisational-social structure around it. Therefore, understanding and supporting this structure, is critical to engineer software better, e.g., fitting dynamic users' needs and their new expectations. This can be useful to, for example, use IT to support governance in global corporations [11].

Nagappan et al. [12] show in practice the influence of organisational structure and other "human" aspects on software quality. This and similar works (e.g. Repenning et al. [13] or Viana et al. [14]) bring evidence that motivates our study of social communities in organisations and the debt (if any) connected to them. This family of studies contributes to social debt by providing evidence of its existence and impact. In addition these studies provide valuable data to identify the orders of magnitude that regulate social debt. Our study is related to results noted by Nagappan et al. [12], in that we confirmed the relation between solid and straightforward software development/operations and well-structured software development social organisations. However, in our case we observed a live organisational and social structure to elicit possible causes and effects for its sub-optimality. While Nagappan et al. establish the causality between organisational structures and software quality, we strived to understand patterns of sub-optimality across said structures, e.g., to allow for preventive action by means of social networks analysis (SNA) [15]. More in particular, we found correlations between sets of organisational-social circumstances and additional cost in software process. Also, we reported the recurrent set of circumstances in which said additional costs occur. While the study by Nagappan et al. serves as motivation and theoretical foundation for our study of social debt, the results in this paper are useful for practitioners embarking on software engineering so that recurrent patterns can be detected and avoided.

Studies on socio-technical congruence, first defined by Cataldo et al. in [16] can support the study of social debt. Socio-technical congruence is the degree to which technical and social dependencies match, when coordination is needed. For example, in [17] Cataldo et al. elaborate socio-technical congruence in formal terms and empirically investigate its impact on product quality. Similar works (e.g. [18]), can be used as starting points to evaluate metrics for social debt. Perhaps socio-technical congruence represents a first rudimental metric for social debt in certain development communities. In our study we did not put any emphasis on measurements, although we offer a rough estimation of the debt we encountered. Our measurement, however, is not based on socio-technical congruence. Nevertheless, the results of our study could be used to evaluate if and how socio-technical congruence can be used as a measurement for social debt and to what degree. Indeed, as part of our results we observed that socio-technical congruence is sufficient to express one third of the possible "debt-effects", i.e., those related to collaboration.

An evolution of the socio-technical stream leads to works such as that of de Souza et al. [19] discuss awareness maintenance mechanisms. These mechanisms are intuitively close to the notion of social debt, since their role is to track and maintain project knowledge with the aim of limiting delays and connected "debt". In relation to this, our work offers means to observe social debt in action and relate it to social constructs and characteristics such as awareness.

Bird et al. in [20] use social-network analysis to investigate coordination among groups of developers with socio-technical dependencies. These lead to using social-networks analysis to elaborate on social debt, studying the very foundations of communities, i.e. their social-network representation. In our work, we did carry some limited form of social-network and organisational analysis. An additional study of our data using socio-technical networks as a construct to structure our data might have revealed additional insights in social debt. This study, however, is out of the scope of this article. Conversely, our results in this paper strive to concretise recurrent set of circumstances into patterns that can quickly be reused in practice to avoid connected nasty consequences.

Works in organisational or socio-technical decision making (e.g. scaling agile methods or offshoring) can provide sample arenas in which social debt emerges. For example, Cusick and Prasad in [21] research the process of understanding if the current organisational layout of a company is performant (or even compatible) with certain decisions (going "offshore" in their case). For more informed decision-making, it is vital to measure the social debt (if any) connected to such decisions. Similarly to [21], many works research the influence of organisational decisions on collaboration and product quality aspects, for example [22,23]. These works support the study of social debt's impact in different scenarios, in terms of end-product quality and evolvability. Our work shares similar goals to the works above in supporting decision-making by means of a social debt framework - to establish the variables around the phenomenon - and recurrent patterns behind its emergence - to avoid or calculate additional costs.

Finally, from a social point of view, many works are indeed related to defining and characterising social debt. First, there is a rich and elaborate body of knowledge concerning social capital, e.g., Meverson in [24]. In Layman's terms, social capital assumes the role of an opposite force to social debt in that it measures and maintains the positive value connected to certain organisational and social structures. In continuum with social capital, social and organisational Labianca et al. in 2008 introduced the notion of social ledger [25], conceptually similar to social debt. Quoting from Labianca et al. "[the effect of] the social liabilities that can result from negative relationships [...] in order to flesh out the entire social ledger [of an organization]".

3 Research design

3.1 Research problem and research question

From sociology literature, quoting from Onions [26]: "social debt of a society represents the set of strained social relationships that emerges as a consequence of debtor-creditor circumstances". In software engineering, the concept can be used, for example, to represent the lack of trust within a community [27] or the degree to which it is immature or unable to tackle a certain development problem. We sought to characterise and study social debt, pivoting around the following research questions: "What are the factors at play around social debt during the software lifecycle? Are there patterns in said factors? Can they be mitigated?"

3.2 Research methodology

3.2.1 Empirical background

The results in this article are based on a study in a large IT service provider (which we call *"Capita"* from now on) for the aviation industry. *"Capita"* has around 3,000 employees in several locations in Germany and around Europe. Also, *"Capita"* controls several offices in 14 other countries.

The context of our investigation is a large software project featuring the Integration of two very different software products (which we call "Integra" from now on). The community of developers we analysed involves two geographically distributed production sites A and B. Both sites are responsible for the implementation of incoming user requests (e.g., new requirements, revised requirements, bug-reports, etc.) and maintenance of the two products to be integrated (which we call RED and GREEN, from now on). The main organisational difference between the two sites, pointed to us when we first started our investigation, is that responsibilities in the remote site are limited to follow what is decided by product managers in Site A. Product managers are responsible for management, software architecture, requirement elicitation and critical decision making. Also, while RED was a well established product, active for well over 10 years, the GREEN product and people were relatively new.

In our study, the objective of *"Capita"* was to: (a) clarify the organisational and social problems for the project under inquiry by distilling a clearer organisational picture (i.e., to understand the organisational scenario involving the two sites and their ramifications to other branches of the *"Capita"*, if any); (b) later on, consolidate and generalise our approach for this study for further reuse in other projects. The gathered data was to be analysed and evaluated, highlighting the pros and cons (consistencies, completeness, weakness, strengths, etc.) from a decision-making point of view, highlighting the roles and factors involved in the different groups taking part in the scenario, and trying to find ways to harmonise their collaboration for the benefit of the Integration project. The industrial partner instructed us to possibly describe how the decision making process should be carried out at best and what knowledge elements should be part of the decision making so that resulting organisational and social

structure is more sound, well constructed and better fit for the development effort at hand.

3.2.2 Data collection

The data we used to obtain our results is based on 16 semi-structured interviews (with an average of 90 mins per interview) as well as 3 focus-groups (with a duration average of 3 hrs) and two workshops (half-day) to investigate the scenario over a period of 6 months. The study involves a total of 22 people, including: managers, architects, developers, operators, Integration engineers, testers, technological assistants, logistic assistants and product owners. Interviewee selection was carried out to ensure complete coverage of the development/operations community at hand. This criterion made sure that the entire organisational and social scenario was investigated. Nevertheless, to access difficult to reach or hidden populations in our scenario, we adopted "Snowball" sampling, i.e., asking our interviewees "who else should we ask about <a topic>?" [28]. This is typical in scenarios such as ours where fear and similar social factors may obscure key people or information from researchers' view.

Interviews[a] were structured according to procedures and guidelines suggested by Neville-Neil [29]. 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 guidelines previously introduced by Morgan et al. [30]. Finally, workshops were structured according to the "Working Group" organisational and social structure in [5]. The aim of the workshops was to provide validation of observations made and refinement where possible, e.g., for unclear or misunderstood concepts. A final workshop was held with all participants present, to further validate our observations after analysis was completed. A report of the study was compiled into a presentation and hand-outs with possible comments. After the presentation, the results were discussed in groups and hand-outs were used for feedback. Following strict non-disclosure agreements, all transcriptions were completely anonymised at the source.

3.3 Analysis methods

The findings presented in this paper were elaborated following a rigorous empirical analysis approach based on several methods and described in the following.

3.3.1 Grounded theory

The results reported in this article were obtained conducting an exploratory case study, according to guidelines proposed in Runeson et al. [31]. We analysed said material using Grounded Theory [32,33]. Our GT approach is structured as follows:

1. **Open Coding** - (4 phases)
   a) Pilot study: a set of 3 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 organised into a hierarchy of codes based on emerging relations between concepts. Thus structured, the start-up list of codes was used to code the rest of the interviews. Each interview transcript was analysed line by line with the list of codes. A code was applied if it reflected a concept in a paragraph, i.e., microanalysis [34].
   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. "Sub-Optimal Organisational Structure" causes "Social Debt", etc.)
   b) Aliasing: the definitions of all concepts coded were compared with each other to identify aliases.
3. **Theoretical Coding** - (3 phases)
   a) Data arrangement: we captured every portion of text that was coded with a code on a table.
   b) Data modelling: the data was represented in a diagram (see Figure 1). 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 analysed and sorted, trying to identify recurrent patterns, underlying relations and hidden meaning. Our observation was aided by standard analysis methods such as weighted frequency analysis (i.e. by analysing 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 modelling.

Finally, coding was carried out by two independent coders to ensure inter-coder reliability. First, the method was applied by a junior researcher that generated the pilot list of possible codes (39 codes). Second, a post-doc researcher re-coded a fresh version of the entire dataset using the pilot list of codes, resolving issues in concordance with researchers involved.

3.3.2 Causality modelling

To represent causality, we applied the 6C model [35]. The 6C model allows to represent empirical causality by relating six variables: **Cause**, meaning the event or

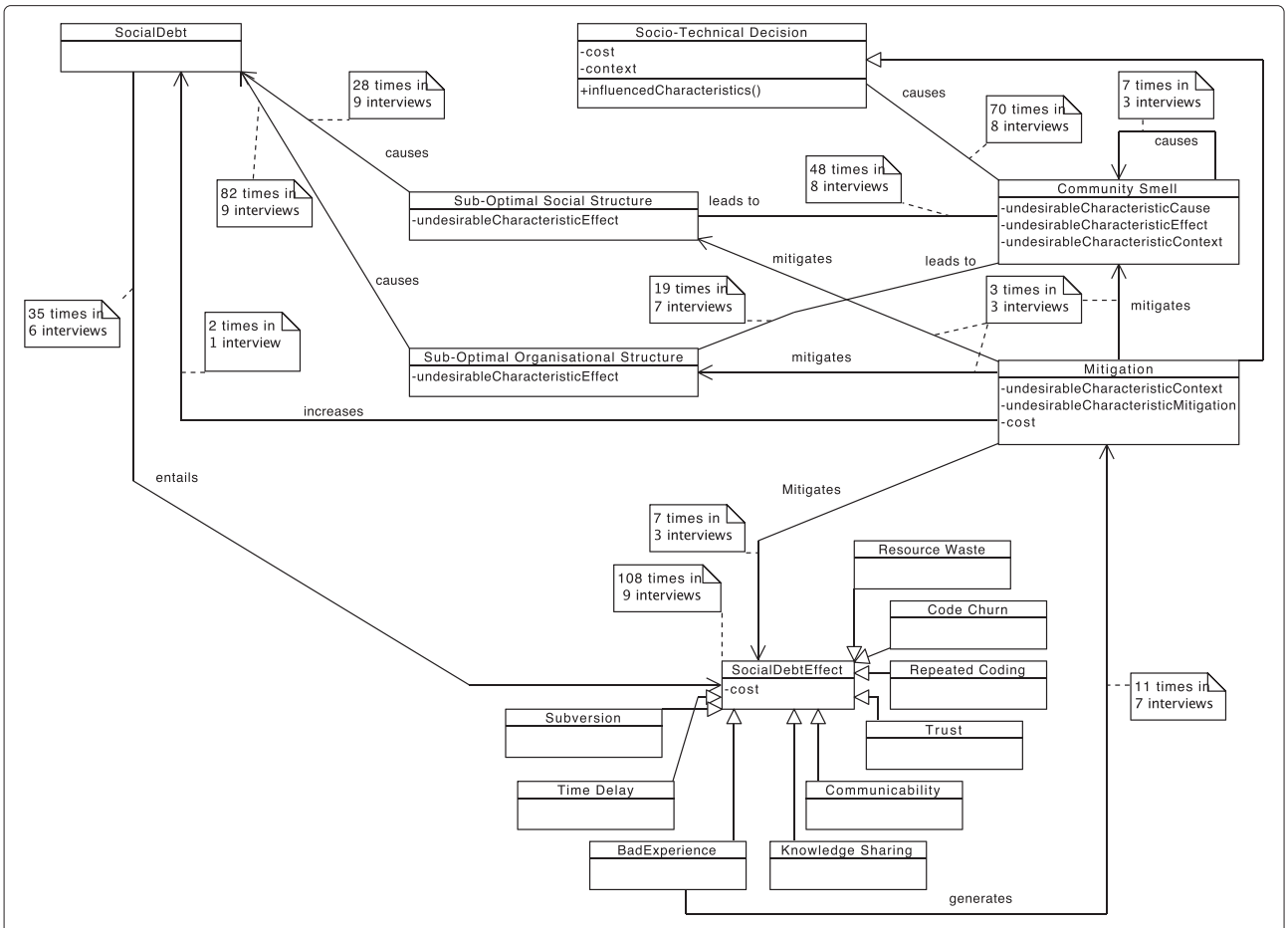

Figure 1 An interpretative framework for social debt.

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; **Contingent**, the event or artefact whose value is compromised by the consequences in the causality function. These six variables are to be found among core concepts.

For example, on Figure 2,"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 cause, the smell

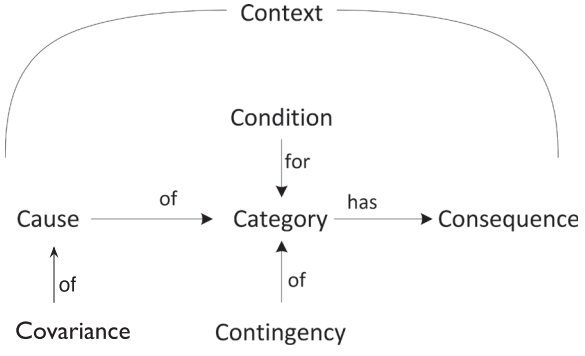

Figure 2 6C model for grounded theory.

"Category" might be subject to a set of conditions (box on top of "Category", see Figure 2). Causes for "Category" exist in the context of the smell (On top of the conditions in Figure 2). Let's assume you might want to do something about the smell by influencing "Category" cause. Certain covariances (or co-evolving 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, we identified existing causality dependencies for about 20% of concepts found. While in most cases the application of the 6C was straightforward, in some cases we fell short in identifying "Covariance" and "Contingent". This is further discussed in Section 5. When our data did not allow us to apply the 6 Cs, we used a " - " sign in the related figure: there was no core concept in our study that was applicable.

4 Results

4.1 Defining social debt: a framework

As previously noted, the core concepts that define social debt emerging from our study are presented in Figure 1 using a UML notation. The framework depicts the core concepts that play a role in social debt (Classes UML Classes). For example, the "SocialDebt" class. The framework also represents the relations behind said concepts (Associations between UML Classes). For example, the "SocialDebt" class is in causal relationship with the "Sub-Optimal Social Structure" class, in particular with its "undesirableCharacteristicEffect" attribute. Comments represent how often the relations were found in the dataset. For example, the causal relationship between the "SocialDebt" class and the "Sub-Optimal Social Structure" class was found 28 times in 9 interview transcripts. To keep the diagram compact, we did not include the occurrence frequency of core concepts[b].

Based on the definitions and relations for concepts in Figure 1, social debt is defined as **a cumulative and increasing cost in the current state of things, connected to invisible and negative effects within a development community. These effects might need some digging in order to be found since they are connected to undesirable, often implicit characteristics in the organisational and social structure [5] emerging in development communities. These characteristics produce an additional cost, e.g., increase the time needed for development.**

To elaborate a (very rough) estimate for the impact of social debt in "Integra", we proceeded as follows: (a) we counted the sub-optimal characteristics reported in its organisational and social structure over a period of 18 months (114 distinct characteristics); (b) we calculated the average delay (4 hrs) using delays directly reported in our interviews for these characteristics, where possible[c]; (c) multiplied the two values.

According to our calculations, This figure equates to a total of 57 days of extra cost connected to social debt. This number is referred to the closing date of our study. Moreover, evidence from the final rounds of interviews and workshops (around the end of our study) suggest that this estimate was bound to increase, e.g., quoting from the last workshop in our dataset: "I am [still] not aware of any way to provide [direct] feedback to an architect. The only way we are using is by talking and asking around [to strangers]".

In addition, our study revealed a number of characteristics of social debt, also evident in Figure 1:

- **Social Debt is indirectly connected to socio-technical decisions; analysing the latter requires aiming for the former** - As evident from Figure 1 there is no direct relation between socio-technical decisions ("Socio-Technical Decision" class in Figure 1) and the emergence of social debt ("SocialDebt" class in Figure 1). For example in our case study of "Integra" we found a key decision to adopt a new programming language as an Integration bridge between RED and GREEN. This decision seemed good at the time it was taken and was motivated by two reasons: (a) the language's efficiency for the job; (b) the language was shared between RED and GREEN, tentatively helping in the creation of an organisational bridge between the two communities. Eventually, however, the decision caused the addition of new technicians to the organisational and social structure which caused fear factors to emerge in the rest of developers. Consequent condescending behaviour resulted in managers and software architects to issue a project slow-down, for problem-solving. At the end of the study, we confirmed that problems related to that decision were still causing slowdowns. This not withstanding, this property might be related to the exploratory nature of our study. Since we introduced social debt only after the interviews, it would have been impossible for interviewees to point out direct relationships between social debt and socio-technical decisions, if any.
- **Social Debt cannot be ascribed to any one software artefact in the development and operations process, rather, it is an emergent property of the development community itself** - As evident from Figure 1, there are no relations between social debt and any software artefacts. Conversely, there is strong evidence that suggests the effect social debt has on software artefacts, as represented by the relation between the "SocialDebt" class in Figure 1) and the "DebtEffect" sub-classes in Figure 1). For example, in our case study of "Integra"

none of the interviewees could point fingers as to who or what was causing trouble (e.g., extra costs or delays) in GREEN or RED and their Integration. There was a perception of problems that pervaded the "Integra" project, and this "hunch" sentiment is constant across our entire dataset. In addition, over 100 "DebtEffects" were reported in 14 interviews as connected to software artefacts as well.

- **Social Debt can be quantified by combining social network analysis of undesirable characteristics with analysing their compounding costs** - As evident from Figure 1 the "SocialDebt" class is strongly correlated to the "Sub-Optimal Social Structure" and "Sub-Optimal organisational Structure" classes. Sub-optimality for these classes is determined by a series of "undesirableCharacteristicEffect" that can be quantified. For example in our case study of "Integra" we found, among others, the following sub-optimal characteristics: "colleague downturn" - people refuse or refrain from helping; "extraneous colleagues" - coworkers do not know each other. Both these characteristics can be measured through social network analysis (e.g., as suggested in [36]); consequently the connected cost may be estimated. This not withstanding, more research is needed to establish and apply the use of SNA for social debt estimation.
- **Social Debt's existence is heralded by organisational and social anti-patterns (i.e., community "smells") which are emerging and recurring across the community** - As evident from Figure 1 the "Socio-Technical Decision" class is in a causal relationship with "Community Smell". Also, according to our evidence, the times at which this causality was reported are antecedent to the emergence of sub-optimal organisational structures. For example in our case study of "Integra" we found a pattern of behaviour recurring in subsequent periods of time and a consequent suboptimal characteristic, "communication delays". We defined this pattern as the "radio-silence" community "smell", that is, a recurring delay (a few hours to half a day) in answering sometimes critical emails or posts. The smell is connected to the previously reported "extraneous colleagues" characteristic. Communication was relying solely on the "kindness-of-strangers" effect. Although the delay may seem small enough, its compounding effect across the community produced a non-trivial delay. An overview of the smells we found is available in Section 4.2.
- **Social Debt can be causing technical debt as well** - For example in our case of "Integra" we found the "replicated-coding" and the "code-churn" technical debt smells [9]. Both smells were emerging from the aforementioned socio-technical decision to adopt a new language as an Integration bridge between the RED and GREEN products as well as a collaboration ground for both communities. As a result of the intended collaboration, people at both ends of the bridge (both in RED and GREEN) replicated pretty much the same functionality (although with different structure) working for re-adaptation of their "end", with consequent waste. Hence, both smells are a "SocialDebtEffect", that is, a negative effect with invisibile socio-technical causes.
- **Social Debt may be mitigated by specific socio-technical decisions ("mitigations")** - For example in our case of "Integra" we found over 50 decisions (or better, instances of "Socio-Technical Decision" class) made to try and "pay back" parts of the accumulated social debt, with mild to very good results. Among these decisions was the decision to adopt a supporting learning community for technicians and staff involved in "Integra". This decision was taken to "pay back" some of the debt connected to the "radio-silence" smell, and succeeded, at least partially. The supporting community increased the social and organisational mesh across the project network and increased mutual social relations such as collaboration, learning and understanding. An overview of the successful mitigations emerging from our study is available in Section 4.3 and mapped to the smells they were addressing.

4.2 Hunches for social debt? Community "smells"!

In essence, community "smells" are sets of organisational and social circumstances with implicit causal relations. These circumstances together are not a problem per se, but if reiterated over time cause social debt, in the form of mistrust, delays, uninformed or miscommunicated architectural decision-making. We found over 70 series of such circumstances in the "Integra" project. 9 such circumstances were recurring over time (i.e., more than 5 times), and reported over the 6-month timespan of our study. Here follow these 9 community "smells":

1. **Organisational Silo effect:** this smell was reported 7 times. In essence, (too) high decoupling development tasks (and related developers) in the community caused lack of communication. Also, high task decoupling lowered developers' mutual awareness [37] and collaboration probably compromised socio-technical congruence [16]. In our scenario these circumstances led to "organisational silos", i.e., sets of loosely dependent development partners wasting resources over the development lifecycle. The *developer silos* smell is shown in Figure 3.

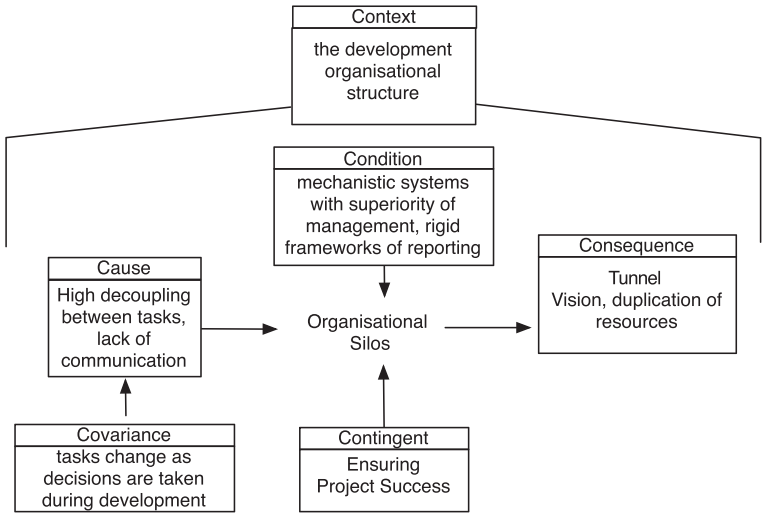

Figure 3 Organisational silo effect, i.e., isolated sub-communities.

Cause: high decoupling between tasks, lack of communication or lack of cooperation in checking task dependencies. *Context:* the entire development and operation community around "Integra". *Conditions:* fine-grained command and control policies for information exchange. Strict superiority management and use rigid frameworks of reporting to control and administer command. *Consequences:* as a consequence of this smell, the community filled with wasted resources (e.g., time) and duplication of code. Also, people in the community reportedly developed a "tunnel vision" with a consequent lack of creativity, lack of cooperation and collaboration. Also, this condition eventually led some developers to make architecture decisions without the necessary background and premises, quoting from our interviews "they [members of team RED] started taking decisions on their own using different format every time, I could see it from TCR (technical change request) documents". *Covariance:* number of tasks and number of dependencies are proportional to architecture decisions - change or increase to such decisions usually has two effects: (a) number of tasks increase; (b) number of dependencies increase. *Contingent:* this smell produces a risk on ensuring project success.

2. **Black-cloud effect:** this smell was reported 8 times. In essence, this smell was caused by two distinct

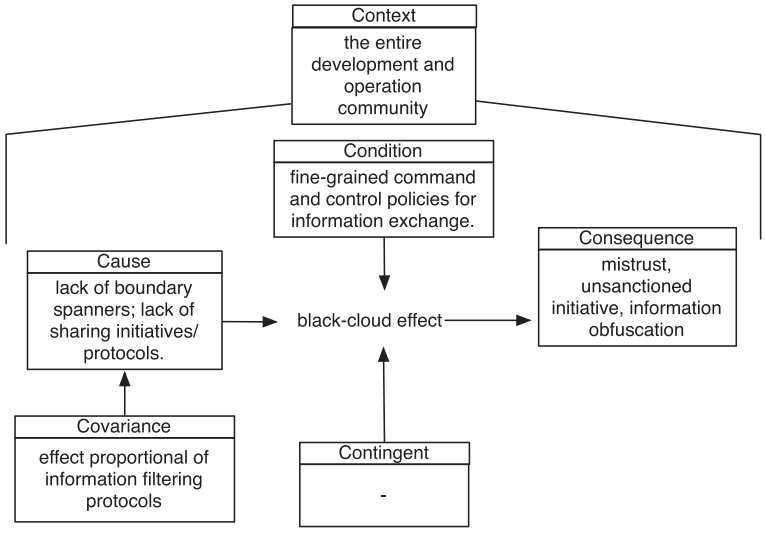

Figure 4 Black-cloud effect, i.e., information overcrowding obfuscates reality.

circumstances taking place at the same time. First, the lack of people able to bridge the knowledge and experience gap between RED and GREEN product teams (i.e., project boundary spanners [38]). Second, the lack of prescribed occasions for knowledge sharing (e.g., daily stand-ups). These two circumstances combined together increasingly created confusion every time knowledge exchange initiatives took place. A "black-cloud" of confusing back-and-forth messages were constantly obfuscating reality. The *black-cloud* effect smell is shown in Figure 4.*Cause:* lack of boundary spanners; lack of sharing protocols. *Context:* the entire development and operation community around "Integra". *Conditions:* fine-grained command and control policies for information exchange. *Consequences:* this smell reportedly created mistrust and people taking "matters and decisions in their own hands", quoting from our interviews. *Covariance:* the effect of this smell is proportional to the number of information filtering protocols (e.g., how many clearance levels) in place to protect information exchange. *Contingent: -*.

3. **Prima-donnas effect:** this smell was reported 9 times. unreceptiveness to change in RED (a legacy product), caused severe isolation problems for people involved. These people become "prima donnas", acting with a seemingly condescending and egotistical behaviour, unable to welcome support from development partners. This compromised the chances of success for "Integra". This scenario led to the emergence of the "Prima Donnas" community smell. Some areas of the development community are still unable or irreceptive towards external influence, cooperation or collaboration. Quoting from the interviews: "everyone wants to pull the stream towards their own windmill and [everyone will vote] for a decision that simplifies their agenda, uncooperative behaviour [...]". The *prima-donnas* smell is shown in Figure 5. *Cause:* innovation and organisational inertia, stagnant collaboration. *Context:* this smell was having an effect in all interactions between RED and GREEN. *Condition: -*. *Consequence:* seemingly egotistical behaviour, lack of communication and collaboration. *Covariance:* organisational changes increased irreceptiveness to external forces, since every change sparked more fear in prima donna sub-communities. *Contingent: - .*
4. **Leftover-techie effect:** this smell was reported 6 times. Increased isolation of maintenance, help-desk and operations technicians in both RED and GREEN reportedly caused technicians to feel as the "last piece of the ladder" and also the "first ones to deal with all complaints [from clients]". This led to creating mistrust some sort of sharing villainy, i.e., misconduct in sharing results or current status in the unwilling technicians. The *leftover-techies* smell is shown in Figure 6. *Cause:* increased isolation between development and operations people. *Context:* this smell was having an effect in all interactions between RED and GREEN as well as the interactions between all products involved and the (paying) customers. *Condition: - . Consequence:* seemingly egotistical behaviour for knowledge and status awareness sharing, lack of communication and general lack of trust. *Covariance:* the more clients the more the negative effect of the smell. *Contingent: - .*
5. **Sharing villainy:** this smell was reported 5 times. Lack of knowledge exchange incentives or face-to-face meetings in "Integra" limited the value

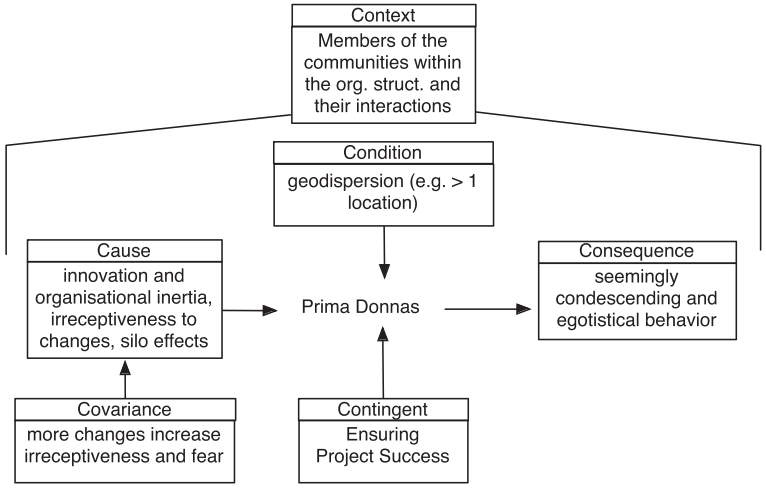

Figure 5 Prima-Donnas, i.e., condescending behaviour due to change irreceptiveness.

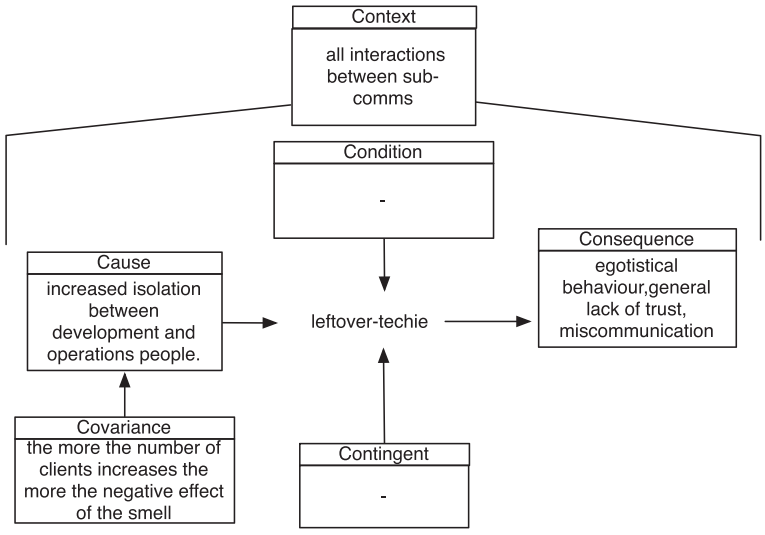

Figure 6 Leftover-techies, i.e., disgruntled operations people.

that developers perceived in sharing their knowledge and experience. Knowledge interaction became a chore, an activity connected more to waste of time and effort, rather than producing evident benefit. This condition limits developers' engagement in knowledge sharing, to a point in which shared information is outdated, unconfirmed or wrong. This scenario leads to the emergence of the "Sharing Villainy" community smell. People do not recognise the importance of sharing good quality knowledge carefully, e.g., by sharing outdated, unchecked updates.

The *sharing-villainy* smell is shown in Figure 7. *Cause:* lack of knowledge sharing incentives as well as activities which promote useful knowledge sharing and synch, e.g., face-to-face meetings. *Context:* this smell negatively affects knowledge exchange and related interactions across the community. For example, people in RED were not motivated enough (e.g., by incentive), enabled (e.g., by buddy-pairing with GREEN) or instructed (e.g., by protocols like agile methods as in GREEN) to communicate with fellow partners overseas, likely disseminate unconfirmed or outdated knowledge, forming the

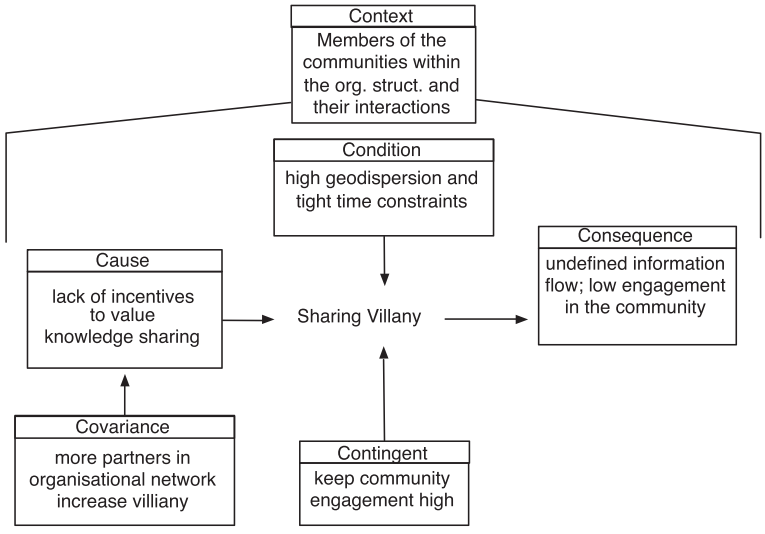

Figure 7 Sharing villainy, i.e., poor quality of knowledge interaction.

ideal conditions for the "Sharing Villainy" smell, as the pattern became recurrent. *Condition: - . Consequence:* undefined information flow and lower engagement in the community, as a consequence of members' detachment. *Covariance: - . Contingent: - .*

6. **Organisational Skirmish:** this smell was reported 7 times. Misalignment between organisational culture in the development unit and the operations unit during "Integra" caused severe managerial issues. For example, different average expertise levels between GREEN and RED reportedly expected dropped productivity up to 50% for weeks at the first attempt to integrate. This scenario leads to an "organisational skirmish" among organisations, each with its own layout and properties. The *Organisational Skirmish* smell is shown in Figure 8. *Cause:* different communication and expertise levels adopted between units involved in the project. *Context:* The skirmish smell was observed during maintenance of RED and GREEN as separate products, but their Integration would likely see the same effect as well. *Condition: - . Consequence:* this smell led to project delay of a few days. *Covariance: - . Contingent: - .*
7. **Architecture hood effect:** this smell was reported 5 times. Architecture decision-makers for "Integra" were far away from both GREEN and RED developers and operators. Also, decisions were taken in an "architecture board" across which it was difficult to find those directly responsible for decisions and their reasoning. This created social strain when decisions were "questioned" by developers or operators in GREEN and RED. In essence, the decision to adopt a software architects' board to speed-up decisions, did in fact increase decision-making but created a "nobody's fault" effect from architects, nobody wanted to take accountability for the decision and lead in its implementation in practice. Also, developers were reportedly blaming the architecture decisions for any technical mishaps during "Integra". The *architecture hood effect:* smell is shown in Figure 9. *Cause:* geographical and socio-technical dispersion of architecture decisions. *Context:* The smell was observed when decisions were being communicated to RED and GREEN. *Condition:* geographical distance between decision makers and others played a major role. *Consequence:* this smell led to uncooperative behaviour across the community. *Covariance: - . Contingent: - .*
8. **Solution defiance:** this smell was reported 9 times. Different levels of experience and different cultural backgrounds divided developers into overly similar subgroups (through homophily). Then, developers divided themselves into factions with completely conflicting opinions concerning socio-technical or technical decisions to be taken. This slowed down "Integra" by up to 2 days and led (in some cases) to "organisational rebellion" (e.g., some developers did not take the decision into account until the last possible minute). The *solution defiance* smell is shown in Figure 10. *Cause:* homophile subgroups. *Context:* The smell was observed when opinions on decisions were being asked to RED and GREEN communities. *Condition:* different experience and cultural backgrounds. *Consequence:* uncooperative behaviour, ignoring decisions. *Covariance:* the more decisions are thrown the more defiance is fostered. *Contingent:* technical debt emerging in the project was reportedly dependent on this circumstance.
9. **Radio-silence:** this smell was reported 16 times. Increasingly formal organisational structure full of

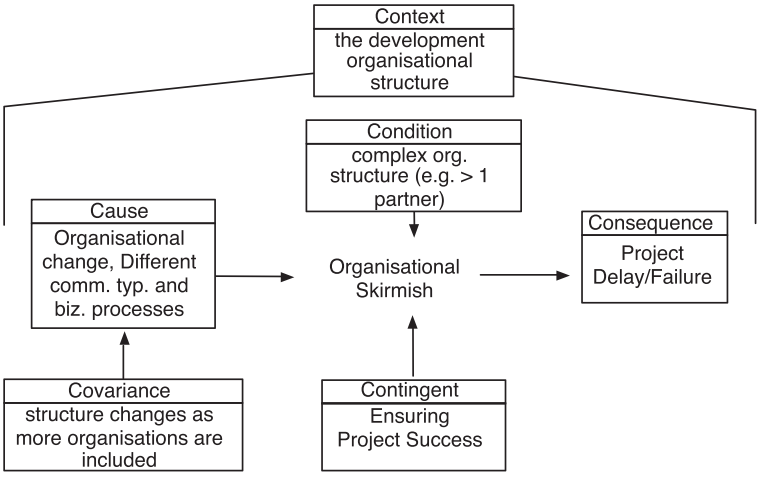

Figure 8 Organisational Skirmish, i.e., clashes of organisational structure.

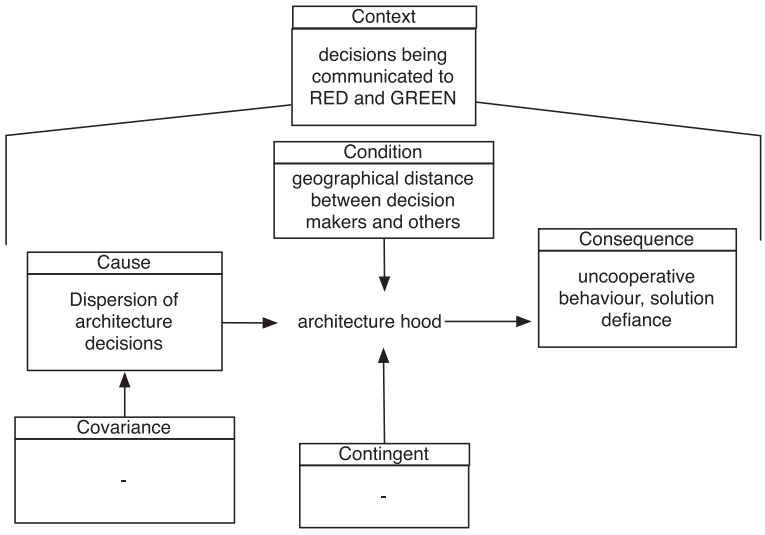

Figure 9 Architecture hood, i.e., architecture decisions are made without accountability, everyone blames the architects.

"regular procedures" forced changes to be retarded, time is lost between people, mostly extraneous between each other, to be notified and certified with. This introduced a fixed recurring delay of 1/2 to 2 days per decision. Also, sometimes people reportedly refrained from asking additional info at all. The *radio-silence* smell is shown in Figure 11. *Cause:* highly formal and complex organisational structure. *Context:* The smell was observed when socio-technical decisions were being communicated to RED and GREEN sub-communities. *Condition:* closed organisational compartments between RED and GREEN. *Consequence:* time delay. *Covariance:* the more decisions are made the more the delay. *Contingent:* -

4.3 It smells! What can I do? Mitigations!

Not all debts in "Integra" were left unpaid or un-tackled. We found over 26 socio-technical decisions operated with the intent of mitigating, although partially, the effect or debt evident in "Integra". Seven interviewees reportedly belonging to the management board of "Integra" reported over 20 socio-technical decisions operated over the span of 18 months to "pay back the debt". In this paper we report

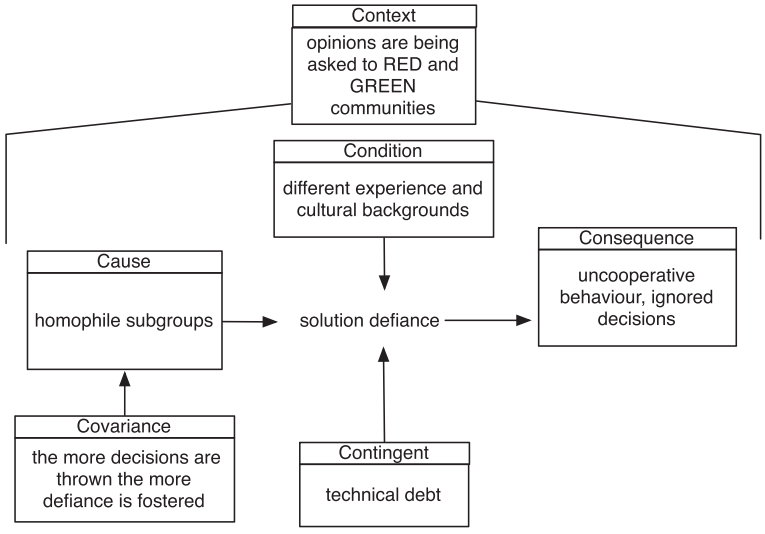

Figure 10 Solution defiance, i.e., overly similar (through homophily) groups with conflicting opinions.

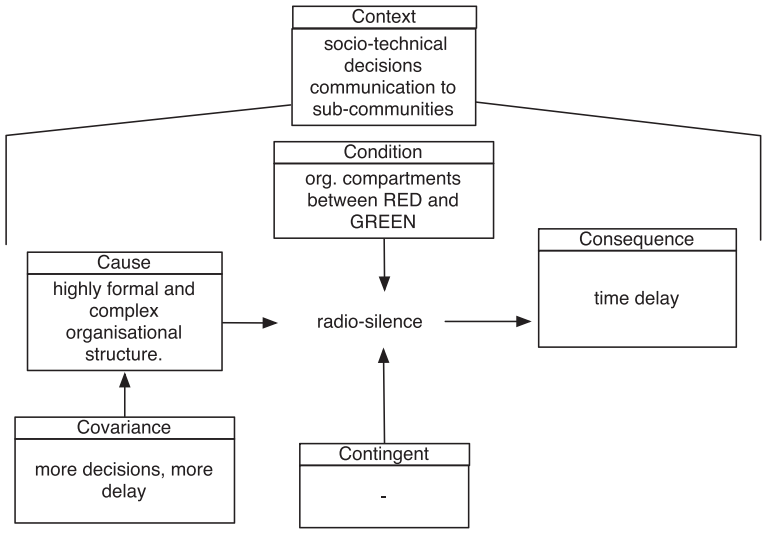

Figure 11 Radio-silence, i.e., too many filters hinder information passing.

only those decisions that reportedly[d] procured a beneficial effect and map these decisions to the relative smell they tried to tackle. Unfortunately, we did not have data to understand to what degree these decisions reduced or hampered the effect of related smells. Also, we found evidence of mitigations that increased social debt rather than the opposite. It is worth mentioning that about 40% of the mitigations adopted did not yield positive outcomes and, in some cases, they made things worse. Also, over 80% of the reported mitigations were generated by "BadExperience" as a malevolent effect connected to social debt (see the relation between "BadExperience" and "Mitigation" in Figure 1).

We found recurring evidence that 6 such decisions procured clear benefits to tackle some community "smells", i.e., "deodorants" for said smells:

1. **Full-circle**: to mitigate the **leftover-techie** smell, managers for "Integra" decided to establish a fixed, dedicated instant-messaging line of communication between operations technicians, managers and key developers, in a manner much consistent with the DevOps trend [39]. This decision indeed greatly hampered further consequences, although not all the connected debt was actually payed back.
2. **Learning-community**: to mitigate the **radio-silence** smell, managers for "Integra" decided to establish an online learning community much similar to the Learning Community type reported in our previous work[5]. All practitioners involved in the project were eventually involved in the community and led by dedicated tutoring and coaching personnel. This decision reduced almost completely further delays and consequences of the radio-silence smell, either directly (by creating stronger social and organisational bonds between practitioners) or indirectly (some practitioners were learning from the community what they would have otherwise asked to stranger colleagues). Nevertheless, we found evidence that subversive behaviour eventually emerged across the community which suggests the need for ad-hoc community structure and management. More in particular, some community members started using the community to "outsource" part of their work or decisions, for example, quoting from our interviews [members of the learning community] are essentially "[different developers from different products and] from different teams each pulling towards their own direction, pushing their own work, instead of finding a common standard [for community structure and goals]".
3. **Culture conveyors**: to mitigate the **prima donnas** and **sharing villainy** smells, managers for "Integra" decided to appoint some developers both in RED and GREEN as architects (i.e., through promotion). The promotion was subject to the condition that new architects would also disseminate a homogeneous organisational culture harmoniously, with the intent of Integrating the RED and GREEN communities as their respective products were being integrated. This decision reduced smell effects but, reportedly, not as much as originally hoped, considered the expense for the promotions. Although we did not have enough data to investigate further, this decision might have,

in fact, payed back some debt while generating some other, connected to the additional introduced cost (promotion scheme).

4. **Stand-up voting**: to mitigate the **architecture-hood** smell managers for "Integra" decided to establish fixed daily stand-ups also for RED (previously adopting a "waterfall" process model). At the end of the stand-ups practitioners would be asked to vote in an anonymous voting-system either for accepting decisions communicated as part of the meeting or proposing feasible alternatives. This decision indeed reduced greatly the effect of the smell almost to none. Nevertheless, the application of this strategy suggests the continued and consistent presence of mistrust among professionals in both RED and GREEN.
5. **Community-based contingency planning**: to mitigate the **prima-donnas** and **solution defiance** smells, managers for "Integra" decided to: (a) make technical (e.g., architecture) and socio-technical (e.g., social or organisational structure changes) together; (b) use the appointed learning community as a device to generate "contingency plans" in case certain technical or socio-technical decisions lead to warning scenarios. This increased the cohesion of the community and reportedly hampered the effects of the smells.
6. **Social wiki**: to mitigate the **prima donnas**, **solution defiance**, **black-cloud effect**, **sharing villainy** and **organisational silo effect** smells, managers for "Integra" decided to adopt a "social wiki" combining practitioners profiles with the artefacts under their care and the connected documentation. This decision was being implemented at the time our study completed its analysis so we are unsure as to the cost or effect of said mitigation. Nevertheless, we decided to report it as it is consistent with the increasing needs for Enterprise 2.0 and enterprise social networking [40].

5 Discussion

This section discusses our findings and their implication. Also, we report a number of lessons learned from the study. These are general insights that might benefit further research into social debt as well as practitioners at hands with it. Finally, we discuss a few threats to validity we identified for this study.

5.1 On the implications of social debt

In the following, the implications for social debt are emphasised in bold.

In previous work [3] we operated a rough comparison of social debt with the notion and state of the art in technical debt. What resulted were a series of key research questions on the overlaps between social and technical debt. Among said questions, was the following: "What decisions cause both technical and social debt?". We indadvertedly stumbled upon the answer to this question, that is, **decisions about using technicalities (e.g. introducing new programming language) to influence community aspects as well, might lead to technical and social debt together**. In our scenario we found 7 such decisions. For example, the previously mentioned decision to use a common programming language between RED and GREEN as a collaboration opportunity. Although it seemed a good decision, well motivated and with sound rationale, the decision ultimately led to technical and social debt. All such decisions should be taken into further inquiry by social and technical debt researchers for further study.

Community smells can be clustered in three sets, depending on their context: (1) smells that exist in a community's structure - i.e., that can be observed and have an effect on the structural properties of a community, such as its formality; (2) smells that exist in the community's context - i.e., that can be observed and have an effect on the constellation of properties surrounding the community's operational environment, such as political boundaries or laws; (3) smells that exist in the community members' interactions - i.e., the set of social and technical relations and actions that allow the community to exist, such as meetings. This suggests a framework to structure future research in discovering community smells, but also suggests an intrinsic difficulty to measure social debt in a precise way, depending on non-trivial relations and characteristics with implicit relations.

The three clusters above share a remarkable similarity with the 3C model for communitarian work, previously defined in [41]. The 3C model states that three basic activities drive the operations for software production: (a) *communication* with peers to realise organisational activities; (b) *coordination* of activities and tasks to achieve planned business goals; (c) *cooperation* on tasks that require concurrent and shared work/expertise. Intuitively, smells that exist in community members' interactions hinder *communication*. Similarly, *coordination*, or organisation in context [42], is compromised by smells that exist in the communities' context. Finally, *cooperation*, is compromised by smells existing in communities' structure.

This intuitive similarity has two key implications: (1) the investigation of social debt [3] needs to focus around the three dimensions in the 3C model and, consequently, rotate around community smells in the corresponding clusters; (2) software engineering research focuses on investigating one out of three sets of smells, that of coordination by means of socio-technical congruence [16] - studying the remaining two dimensions could prove valuable to govern development communities, maximising their social worth or managing their social debt [3], if any.

Influencing "smells" without preemptive study is ultimately a trial-and-error exercise. In our case, we found over 10 relatively uninformed socio-technical decisions made to tackle "smells". These did not yield the desired effect and, in 3 cases, reportedly led to worse outcomes. This suggests that a more accurate study of "smells" and mitigations is in order. This study might reveal previously reported "smells" as well as effective ways in which they were tackled under certain conditions. This study might start from the body of literature in software engineering success or failure or, in parallel, from the body of literature in organisations and social networks research.

5.2 Lessons learned

There are several lessons we learned from studying the scenario at hand.

Integrating two products means joining two (or more) communities as well. In our case, many social, organisational and technical decisions were taken along the way to "merge" the two communities involved into one. However, we discovered a number of "ancillary" people, part of other business units, not directly involved in "Integra" that actually played a role in the project (e.g., helpers, information conveyors, etc.). With this we learned that the organisational and social Integration between communities responsible for a certain product is critical and should be carefully planned and orchestrated.

Software practitioners generally associate guilt and fear to the social and organisational investigation of their efforts. We observed this circumstance in many previous studies of similar nature. It may well be connected with the "my boss will not be happy about this" social dynamic, which may cause the connected anxiety. Nevertheless, with this we learned that the tools with which a development community should be studied or "influenced" should be automated and non-invasive, observing people and software artefacts together as well as based on gamification schemes and similar technology.

Many "Contingent" factors for reported smells were missing. We observed this circumstance on 4 out of 9 reported smells. This can suggest that, for instance, the smells are applicable to multiple possible "Contingents" or "Contingents" for negative effects observed were not addressed by practitioners in the first place. This calls for further research to identify and study "Contingent" factors.

Divide-and-conquer may no longer be fit to rule. In our set of 90 organisational, socio-technical or technical decisions, over 80% of these decisions applied a classic software engineering innuendo: *divide-et-impera*. Most of these decisions, however, implied divisions in the product as well as corresponding divisions in the community. While the former may procure benefits, the latter may compromise everything. With this we learned that a more communitarian formula should be researched for teaching and practicing software engineering. In fact, this observation is corroborated by looking more deeply at the mitigations reported in Section 4.3. All the mitigations reported have the goal of increasing the diversity, connectedness and awareness [43] of a community of people. More research is needed to establish the ways and practices in this theoretical formula. Such research would very well benefit from the study of successful (and failing) open-source communities. These are a clear example of efficient and powerful self-organisation.

5.3 Threats to validity

Based on the taxonomy in [44], there are four potential validity threat areas, namely: external, construct, internal, and conclusion validity.

External Validity concerns the applicability of the results in a more general context. Being this study performed in one organisation, results could be specific to its context. To reapply results and possibly confirm the validity of this study we are planning additional independent exploratory case-studies.

Construct Validity and *Internal Validity* concern the generalisability of the constructs under study, as well as the methods used to study and analyse data (e.g. the types of bias involved). To mitigate these threats, our methods were tailored to use multiple triangulation of data sources. A representative from *"Integra"* verified our interpretations of the data and provided clarifications and corrections where necessary. Partial results and incremental analysis was conducted to gather constant independent feedback by three senior researchers.

Conclusion Validity concerns the degree to which our conclusions are reasonable based on our data. Our conclusions were drawn by an analysis of empirical evidence using known and confirmed methods from literature such as, coding, gap- and taxonomy analysis. The approach and instruments that we used to gather such evidence were presented and validated in previous work [2,4,45].

6 Conclusions and future work

In this paper we discuss and elaborate on the notion of social debt that was originally introduced in [3]. To investigate social debt in action we conducted a case study in industry. To answer our primary research questions, namely, "What are the factors at play around social debt during the software lifecycle? Are there patterns in said factors? Can they be mitigated?", this article offers four key contributions.

First, a framework to define and interpret social debt. This can be used in practice to identify the variables that govern social debt, e.g., for more informed decision-making.

Second, a list of community "smells". These are socio-technical anti-patterns that may apparently look normal but reflect unlikeable community characteristics, such as anti-social organisational behaviour across the community, e.g., developers that refuse or delay information sharing. These patterns can be used in practice to "sense" the precursors of social debt in much the same way code smells fathom technical debt.

Third, a list of mitigation strategies to community "smells", as emerging from our case study. These can be used in practice to reduce the negative effects connected to some "smells".

Fourth, a list of lessons learned as part of our case study that further discuss social debt and its implications. These can be used by researchers interested in pursuing further social debt research.

From our study we learned that social debt is a force to be reckoned with, hand in hand with technical debt. This force is connected to sub-optimal characteristics in software development communities. Further research is needed to generate a more systematic approach. For example, the use of community detection and characterisation mechanisms via social networks analysis might be used to understand if social debt can somehow be measured automatically. Also, investigating further the relation between software architectures and "organisational architectures" might reveal patterns that lead to social and technical debt.

In the future, we plan to devise mechanisms to visualise and study the social community structure of development communities, by putting together their socio-technical properties and observable characteristics. In so doing, we hope to discover ways to mine data from software development communities that can lead to the discovery of their sub-optimal characteristics, and, possibly, any connected "smells". Also, we plan to elaborate further on the notion of community "smells" perhaps starting from analysing literature in software engineering (e.g., studying software failure stories). More in particular, we plan to answer the following research questions:

"What software engineering artefacts are affected by community smells?" Much information and data in our dataset concerning software artefacts (e.g., software architecture, requirements, integration requests, etc.) in our investigation scenario remains to be analysed - perhaps this data can be analysed in continuation with the study in this paper, e.g., to understand if community smells have a direct or indirect impact on software itself.

"How can social debt be measured?" Social debt clearly entangles a number of software and lifecycle artefacts, finding measurements for the negative characteristics connected to social debt in said artefacts is the starting point to measure social debt in the first place. This research venue might benefit from research in Mining Software Repositories (MSR) since it involves mining software products to establish causality. However, an exploratory study is needed to establish what should be measured and how.

"Are there community smells previously reported in literature?" Software engineering literature offers empirical research in software failure that could potentially contain a number of community smells. Mapping this portion of software engineering literature could be a valuable tool for practitioners embarking on complex software engineering endeavours involving more than one team or organisation.

7 Endnotes

[a] Interview guide is only available through written request.

[b] Nevertheless, every core concept was found more than 48 times in the entire dataset.

[c] The average of 4 hrs was calculated on 34 sub-optimal characteristics for which our interviewees explicitly gave us a delay - this average was then extended to all sub-optimal characteristics found.

[d] By at least two interviewees.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
DAT carried out the analysis and wrote a draft report to be submitted to the attention of PK, PL and HVV. DAT also drafted a publishable manuscript and submitted it to all coauthors. All authors helped partially to analysis and contributed during brainstorming and theory-building. HVV and PK provided major edits to the manuscript while PL contributed to study design, provided major edits to the manuscript and co-located editing, reviewing and analysis sessions via Skype. All authors read and approved the final manuscript.

Acknowledgements
The authors would like to thank Dr. Francesco Castri for his invaluable contribution to this study.

Author details
[1] VU University Amsterdam, Amsterdam, The Netherlands. [2] University of British Columbia, Vancouver, Canada.

Received: 29 October 2014 Accepted: 5 March 2015
Published online: 04 May 2015

References

1. Keyes J (2011) Social Software Engineering. Taylor & Francis, Auerbach Series, Boca Raton, FL
2. Tamburri DA, di Nitto E, Lago P, van Vliet H (2012) On the nature of the GSE organizational social structure: an empirical study. doi: 10.1109/ICGSE.2012.25
3. Tamburri DA, Kruchten P, Lago P, van Vliet H (2013) What is social debt in software engineering?. In: Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop On, Washington, DC. pp 93–96. doi:10.1109/CHASE.2013.6614739
4. Tamburri DA, Lago P, van Vliet H (2013) Uncovering latent social communities in software development. IEEE Software 30(1):29–36. doi:10.1109/MS.2012.170
5. Tamburri DA, Lago P, van Vliet H (2013) Organizational social structures for software engineering. ACM Comput Surv 46(1):3

6. Capek PG, Frank SP, Gerdt S, Shields D (2005) A history of ibm's open-source involvement and strategy. IBM Syst J 44(2):249–258
7. Kruchten P, Nord RL, Ozkaya I (2012) Technical debt: From metaphor to theory and practice. IEEE Software 29(6):18–21
8. Cunningham W (1993) The WyCash portfolio management system. OOPS Messenger 4(2):29–30
9. Zazworka N, Shaw MA, Shull F, Seaman C (2011) Investigating the impact of design debt on software quality. In: Proceedings of the 2nd Workshop on Managing Technical Debt. MTD '11. ACM, New York, NY, USA. pp 17–23. doi:10.1145/1985362.1985366. http://doi.acm.org/10.1145/1985362.1985366
10. Conway ME (1968) How do committees invent. Datamation 14(4):28–31
11. Wenger E, McDermott RA, Snyder W (2002) Cultivating Communities of Practice: a Guide to Managing Knowledge. Harvard Business School Publishing
12. Nagappan N, Murphy B, Basili V (2008) The influence of organizational structure on software quality: an empirical case study. In: International Conference on Software Engineering. IEEE, Leipzig, Germany. pp 521–530
13. Repenning A, Ahmadi N, Repenning N, Ioannidou A, Webb D, Marshall K (2011) Collective programming: making end-user programming (more) social 6654:325–330. http://www.bibsonomy.org/bibtex/26e8152bfef95458d1dcec728a4f51c2a/dblp
14. Viana D, Conte T, Vilela D, de Souza CRB, Santos G, Prikladnicki R (2012) The influence of human aspects on software process improvement: Qualitative research findings and comparison to previous studies. In: EASE. pp 121–125
15. Meneely A, Williams L, Snipes W, Osborne JA (2008) Predicting failures with developer networks and social network analysis. In: Harrold MJ, Murphy GC (eds). SIGSOFT FSE. ACM. pp 13–23
16. Cataldo M, Herbsleb JD, Carley KM (2008) Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '08. ACM, New York, NY, USA. pp 2–11. doi:10.1145/1414004.1414008. http://doi.acm.org/10.1145/1414004.1414008
17. Cataldo M, Mockus A, Roberts JA, Herbsleb JD (2009) Software dependencies, work dependencies, and their impact on failures. IEEE Trans Software Eng 35(6):864–878
18. Kwan I, Schroter A, Damian D (2011) Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. IEEE Trans Software Eng 37(3):307–324. doi:10.1109/TSE.2011.29
19. de Souza CRB, Redmiles DF (2011) The Awareness Network, To Whom Should I Display My Actions? And, Whose Actions Should I Monitor? IEEE Trans Software Eng 37(3):325–340
20. Bird C, Nagappan N, Gall H, Murphy B, Devanbu P (2009) Putting it all together: Using socio-technical networks to predict failures. In: Proceedings of the 2009 20th International Symposium on Software Reliability Engineering. ISSRE '09. IEEE Computer Society, Washington, DC, USA. pp 109–119. doi:10.1109/ISSRE.2009.17
21. Cusick JJ, Prasad A (2006) A practical management and engineering approach to offshore collaboration. IEEE Software 23(5):20–29
22. Jaktman CB (1998) The influence of organisational factors on the success and quality of a product-line architecture. In: Australian Software Engineering Conference. IEEE Computer Society, Washington, DC. pp 2–11
23. Andreou AS (2003) Promoting software quality through a human, social and organisational requirements elicitation process. Requir Eng 8(2):85–101
24. Meverson EM (1994) Human capital, social capital and compensation: The relative contribution of social contacts to managers' incomes. Acta Sociologica 37(4):383–399
25. Labianca G, Brass DJ (2006) Exploring the social ledger: Negative relationships and negative asymmetry in social networks in organizations. Acad Manage Rev 31(3):596–614
26. Muir DE (1962) The social debt: An investigation of lower-class and middle class norms of social obligation. American Sociological Review 27(4):532–539
27. Moe NB, Smite D (2008) Understanding a lack of trust in global software teams: a multiple-case study. Software Process: Improvement and Practice 13(3):217–231
28. Atkinson R, Flint J (2012) Accessing hidden and hard-to-reach populations: Snowball research strategies 33
29. Neville-Neil G (2011) Interviewing techniques. ACM Queue 9(6):30
30. Morgan DavidL., Krueger RichardA. (1993) 1. In: Morgan DL (ed). When to use focus groups and why. SAGE Publications, London. pp 3–19. http://www.bibsonomy.org/bibtex/2ce683d8f889e49e740d81683fdc2c2b9/pkraker
31. Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Eng 14(2):131–164
32. Corbin J, Strauss A (1990) Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative Sociology 13(1):3–21
33. Schreiber C, Carley KM (2004) Going beyond the data: Empirical validation leading to grounded theory. Comput Math Organization Theory 10(2):155–164
34. Onions PEW (1962) Grounded theory applications in reviewing knowledge management literature. 1–20
35. Glaser BG (1978) Theoretical Sensitivity: Advances in the Methodology of Grounded Theory. Sociology Press, San Francisco, CA
36. Kilduff M, Tsai W (2003) Social Networks and Organizations. SAGE Publications Ltd, London
37. Chisan J, Damian D (2004) Towards a model of awareness support of software development in gsd. IEE Seminar Digests 2004(912):28–33. doi:10.1049/ic:20040309
38. Peng Y, Sutanto J (2012) Facilitating knowledge sharing through a boundary spanner. IEEE Trans Prof Commun 55(2):142–155
39. Labs P (2013) 2013 state of devops report. Technical Report. Available Online
40. Li M, Chen G, Zhang Z, Fu Y (2012) A social collaboration platform for enterprise social networking. In: Gao L, Shen W, Barths J-PA, Luo J, Yong J, Li W, Li W (eds). CSCWD. IEEE, Washington, DC. pp 671–677
41. Swart J, Henneberg SC (2007) Dynamic knowledge nets - the 3c model: exploratory findings and conceptualisation of entrepreneurial knowledge constellations. J Knowledge Manage 11(6):126–141
42. Boella G, van der Torre L (2006) Coordination and organization: Definitions, examples and future research directions. Electron Notes Theor Comput Sci (ENTCS) 150(3):3–20
43. Manteli C, van der Hooff B, van Vliet H (2014) The Effect of Governance on Global Software Development: An Empirical Research in Transactive Memory Systems. Inf Software Technol 56(10):1309–1321
44. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers, Norwell, MA, USA
45. Tamburri DA (2012) Going global with agile service networks. IEEE Conference Proceedings, Washington, DC