Received: 2 September 2019 | Revised: 14 May 2020 | Accepted: 1 June 2020

DOI: 10.1002/spe.2874

RESEARCH ARTICLE

WILEY

"The Canary in the Coal Mine..." A cautionary tale from the decline of SourceForge

Damian Andrew Tamburri[1] | **Kelly Blincoe**[2] | **Fabio Palomba**[3] | **Rick Kazman**[4]

[1]Jheronimus Academy of Data Science - Data and Engineering Lab, Hertogenbosch, the Netherlands

[2]Department of Electrical, Computer, and Software Engineering, University of Auckland, Auckland, New Zealand

[3]Department of Computer Science, University of Salerno, Fisciano, Italy

[4]Department Information Technology Management, University of Hawaii, Honolulu, Hawaii, USA

Correspondence
Damian Andrew Tamburri, Jheronimus Academy of Data Science, 's-Hertogenbosch, the Netherlands.
Email: d.a.tamburri@tue.nl

Funding information
H2020 European Institute of Innovation and Technology, Grant/Award Numbers: 825040, 825480

Summary

Forges are online collaborative platforms to support the development of distributed open source software. While once mighty keepers of open source vitality, software forges are rapidly becoming less and less relevant. For example, of the top 10 forges in 2011, only one survives today—SourceForge—the biggest of them all, but its numbers are dropping and its community is tenuous at best. Through mixed-methods research, this article chronicles and analyze the software practice and experiences of the project's history—in particular its architectural and community/organizational decisions. We discovered a number of suboptimal social and architectural decisions and circumstances that, may have led to SourceForge's demise. In addition, we found evidence suggesting that the impact of such decisions could have been monitored, reduced, and possibly avoided altogether. The use of sociotechnical insights needs to become a basic set of design and software/organization monitoring principles that tell a cautionary tale on what to measure and what not to do in the context of large-scale software forge and community design and management.

KEYWORDS

automated architecture analysis, community analysis, forge design, software failure

1 | INTRODUCTION

Over the years the open source software movement has contributed to a dramatic reduction of software costs and release times, while increasing general quality.[1] At the same time, the way in which open source software is built has changed radically, from the domain of just a few, to an enormous economic force. Now entire open source software ecosystems have become subject of flourishing research and practice.[2,3] However, one of the phenomena that played a role in this change is the failure of software forges. A software forge provides a platform for hosting software projects and usually offer code hosting and bug tracking. While many software forges existed in 2011, only one seems to still be in existence: SourceForge.[1]SourceForge was the biggest of all forges by count of projects, commits, and participants, but these numbers have dramatically declined in recent years.

[1]Link: http://sourceforge.net/. Note that other more timely platforms like GitHub provide additional collaboration tools and enable various social network analyses. They are, therefore, more advanced than classical forges; this is the reason why we do not consider them as forges.

In this study, we aim to understand the shape and indicators on the *Canary* in the *Coal Mine*, namely, we examine potential early indicators of trouble in SOURCEFORGE using mixed-methods empirical software engineering research.[4] To do this, we adopted a mixed-method research approach, by (1) conducting interviews to learn about team members' perceptions on the project and community health during the duration of the project and (2) analyzing 9 years of commit activity and 4 years of issue reports and mailing lists. Our analysis subjects are SOURCEFORGE itself, and its supporting software infrastructure.

Members of the original team perceived various social and technical issues that contributed to the decline of SOURCEFORGE. Paired to that, in the quantitative analysis, we also observed a lack of organizational stability in SOURCEFORGE. This instability occurred at the same time as an increasing number of changes to the code base that resulted in huge increases of technical debt whose introduction may be due to the lack of communications/coordination among the contributors. Thus, a variety of social and technical factors contributed to the downfall of SOURCEFORGE and its supporting infrastructure.

Structure of the article. Section 2 describes the research setting and overviews the research methodology adopted to address the posed research questions. Section 3 reports our cautionary tale on SOURCEFORGE and ALLURA and its supporting infrastructure. In Section 4, we discuss the key findings and lessons learned from our study, while Section 5 summarizes the possible threats to the validity of the study and how we mitigated them. Section 6 discusses the related literature on the topic; finally, Section 7 concludes the article.

2 | METHODOLOGY

2.1 | Research setting

This study investigates SOURCEFORGE and its supporting software infrastructure, the APACHE ALLURA.[2] SOURCEFORGE was created by VA Software and was first launched in 1999. It was one of the first to offer free code hosting for open source software projects, a revolutionary service at the time. It offered free access to concurrent versioning system, a bug tracker, and mailing lists to open source projects. For many years, it was the biggest open source software development and collaboration website.[5,6]

In 2009, SOURCEFORGE initiated the APACHE ALLURA platform. ALLURA is *"an Open Source, extensible, web-based platform that provides integrated software tools for collaborative software development*.[7]" ALLURA provides an integrated issue tracker, built-in discussion forums, a code repository, and more. It was submitted to the Apache Incubator in 2012 and became a Top-Level Project in 2014.[7] Originally, the project stems from the codebase that sustained developer tools for SOURCEFORGE (in PHP) and was redesigned using leaner programming languages and frameworks such as Python. Although it is best known as being the platform behind SOURCEFORGE, ALLURA also powers several software-intensive platforms such as the Open Source Projects Europe [3], the DLR German Aerospace Center [4], and DARPA's VehicleForge.[5]

2.2 | Research problems and questions

In this study, we aim to understand how sociotechnical factors contributed to the decline and downfall of SOURCEFORGE. We do this by considering the team member perceptions around the decline of SOURCEFORGE and examining the social and technical structures of the APACHE ALLURA software community, which is responsible for the development of SOURCEFORGE. We formulate three research questions:

- **RQ1.** *What were the perceptions of the SOURCEFORGE team on the project and its community health?*
- **RQ2.** *Is there evidence of problems in the community and technical structures of SOURCEFORGE?*
- **RQ3.** *Is there evidence of architectural problems in SOURCEFORGE?*

[2]https://allura.apache.org/
[3]https://opensourceprojects.eu/
[4]http://www.dlr.de/dlr/en/desktopdefault.aspx/tabid-10002/
[5]https://cps-vo.org/group/avm/vehicleforge

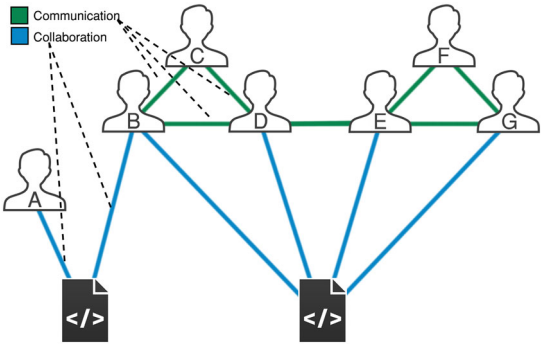

FIGURE 1 High-level representation of the developer social network structure we used for community smell detection showing both communication and collaboration graphs [Colour figure can be viewed at wileyonlinelibrary.com]

To answer **RQ1**, we interviewed members of the SOURCEFORGE team to learn about their perceptions on the project and community health during the duration of the project [6]. To answer **RQ2**, we analyzed 9 years of commit activity of the SOURCEFORGE/ALLURA project to identify the evolution of the source code and its structure. We also analyzed the last 4 years worth of data describing its community and organizational structure. To answer **RQ3** we analyzed the architecture of eight versions of APACHE ALLURA, focusing on its decoupling level (DL) and its architectural flaws.

2.3 | Interview data

We interviewed four members of the SOURCEFORGE team. Our participants covered 25 years of SOURCEFORGE history. The participants were SOURCEFORGE developers, managers and analysts. Participants included junior (<2 years experience), senior (3-5 years experience), and expert (6+ years experience) members, in terms of SOURCEFORGE expertise. Our participants had varied backgrounds including business informatics, software operations, software engineering, management, and digital IT marketing (Figure 1).

2.4 | Contribution and communication data

The APACHE ALLURA software repository was mined in its entirety by using the CODEFACE tool.[8] We elicited all 4 years worth of data for ALLURA about: (i) file change quantities and commit statistics; (ii) total quantity of changes over time; (iii) commit-message sizes and page-ranking of contents. A total of 8902 commits were analyzed. Finally, in terms of communication data, we scraped ALLURA issue-tracking and mailing lists, computing 3-month community structure snapshots from both sources and for all the 4 years of data currently available for ALLURA.[7] A total of 32 community sociograms[9] were generated, visualized, and analyzed using time-series analysis.

A discussion of the analyses conducted on the above data is contained in the respective results sections for each research question (see Sections 3.1, 3.2, and 3.3).

2.5 | Operationalization and data mining

To attain our results, we exploited previous work in community analysis and sociotechnical measurement of software development networks.[10,11] Both works report, respectively, on the operationalization and reapplication of community smells and other relevant sociotechnical metrics (see table 4 from Palomba et al[11]) for the qualities of software processes and products. Furthermore, both works reflect extensions to the well-known CodeFace tool, a Siemens tool for application lifecycle intelligence originally introduced by Joblin et al.[8] More specifically, the following operationalization is used in the scope of this work (tailored from Palomba et al[10,11]).

[6] interviewees kindly asked to share interviews selectively and therefore, interview transcripts are available upon written request.
[7] As an example, ALLURA mailing lists are available here: http://mail-archives.apache.org/mod_mbox/allura-dev/

Starting from the developer networks built by CODEFACE, we detect instances of smells according to the formalization below. For all of them, a premise is needed:

Premise. Let $G_m = (V_m, E_m)$ be the communication graph of a project and $G_c = (V_c, E_c)$ its the collaboration graph.

More precisely, for *communication* we mean the relation by which two or more developers communicate with each other through any channel: for example, a communication link between two developers is established in case they reply to the same discussion within a mailing list or they comment on the same issue in the issue tracker. As for *collaboration*, we mean the relation for which two or more developers have worked on the same source code elements. This is established by considering the change history of a project, looking for cases where two or more developers have modified the same code entities.

2.5.1 | Organizational silo effect

The organizational silo effect occurs when the developers break into isolated subcommunities with little or no coordination between the subcommunities.[12,13] That is, there are two subteams that cannot properly communicate with each other. In the communication graph this manifests itself as two (relatively isolated) subgraphs with just one or two people connecting them. With the occurrence of organizational silo effects, social debt manifests as decaying communication across subcommunities and consequent negative effects on developers' situational awareness[14] as well as degradation of projects' sociotechnical congruence.[12,15] In addition, according to recent findings,[12] the organizational silo effect may lead to tunnel-vision, since participants may focus their cooperation and communication solely on other members of their narrow subcommunity rather than on the broader community. Finally, community members belonging to an *organizational silo* may exhibit egotistical behavior leading to unsanctioned architectural decisions[16] as well as defiance of the decisions of others.[12]

For the sake of precision, we capture the organizational silo effect at the finest grain possible, that is, that of collaboration dyads: pairs of cocommitting developers. An example is shown in Figure 2. Here, the organizational silo effect is reflected on developer "1," who does not communicate with developer "2" even though "1" is collaborating with "2." Conversely, developer "2" is communicating with (at least) one other developer, "3," who belongs to a subcommunity other than "1." Considering the example proposed in Figure 1, an operationalization of the identification pattern for the organizational silo effect has two steps. In the first step, the identification mechanism compares the collaboration network (bottom half of Figure 1) with its communication counterpart (top half of Figure 1). Then it verifies that the developer identified by the letter A is present in the collaboration network, that is, A commits to files cocommitted by others, but is not present in the communication developer social network (DSN) reflecting those files.

2.5.2 | Lone wolf effect

The *lone wolf* community smell reflects circumstances in which communication may indeed be present but insufficiently addressing project needs.[12,13] The result is developer free-riding and unsanctioned architectural decisions that cause nasty

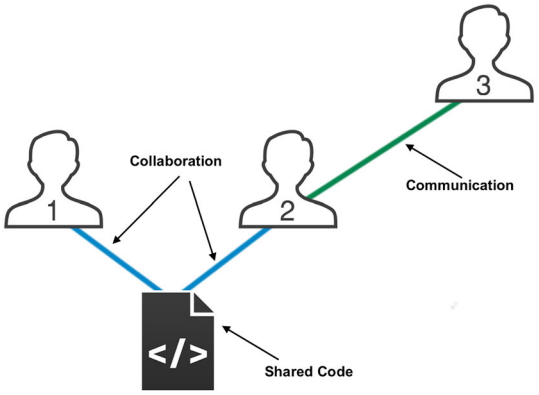

FIGURE 2 *Organizational silo* effect community smell identification pattern [Colour figure can be viewed at wileyonlinelibrary.com]

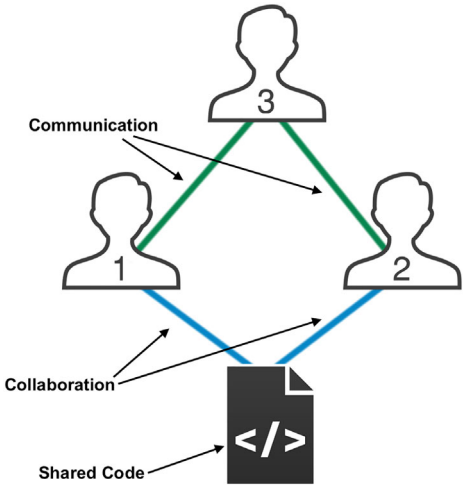

FIGURE 3 Lone wolf community smell identification pattern [Colour figure can be viewed at wileyonlinelibrary.com]

ripple effects such as code duplication and churn.[14] Thus, we define the set of *lone wolf* pairs L as the set of collaborators that do not directly or indirectly communicate with each others.

The identification pattern for the *lone wolf* smell is based on the detection of development collaborations between two community members that have intermittent communication counterparts or feature communication by means of an external "intruder," that is, not involved in the collaboration. A simple example is given in Figure 3. In this example two developers, "1" and "2," are collaborating on some code, but they are not connected by any communication link other than developer "3," who is not cocommitting on a shared file. In this case, either developer "1" or developer "2" (or both) can develop a *lone wolf* community smell.

This smell reflects the presence of possible side effects generated by the *organizational silo* such as communication decay or negative influence on developer awareness and heavy sociotechnical congruence degradation. Our conjecture is that the occurrence of the *organizational silo* effect is not negative per se. But when that occurrence is compounded by the occurrence of *lone wolves*, extra attention must be paid to avoid negative consequences such as delays and unmanageable social debt. The *lone wolf* smell reflects dyads of cocommitting (collaborating) software developers who exhibit uncooperative behavior and mistrust by not appropriately communicating.

2.5.3 | Black cloud effect

The black cloud effect leads to negative social interactions within a software development community featuring: (i) community members' inability to cover knowledge and experience gaps between two different software products developed within the same software community;[12] (ii) lack of periodic and official opportunities to share and exchange knowledge between all community members.[12,13] Whenever these two circumstances occur together, they can generate a "black cloud" of misinformation (eg, confusing, delayed, or unnecessary communication that generates communication overload) within the community. The main consequence of the *black cloud* effect is to obfuscate project vision, compromising progress.[12,13] The occurrence of this community smell can be generated or worsened by several sociotechnical triggers:

- absence of ad hoc protocols for knowledge sharing;
- lack of boundary spanners;
- presence of inefficient communication filtering protocols.

Moreover, the *black cloud* effect smell is associated with several other side effects such as: lowering of trust between developers, information obfuscation, as well as inception of the *organizational silo* effect, due to the rise of egoistic behavior. The identification pattern for the *black cloud* effect smell reflects subcommunities that in subsequent release windows do not communicate, with the exception of two community members (ie, boundary spanners in social-network jargon[17]), one belonging to each subcommunity. The detection of the *black cloud* instances starts with the identification of vertex clusters as already implemented in CODEFACE. More specifically, let $P = \{p_1, \ldots, p_k\}$ be a mutually exclusive and

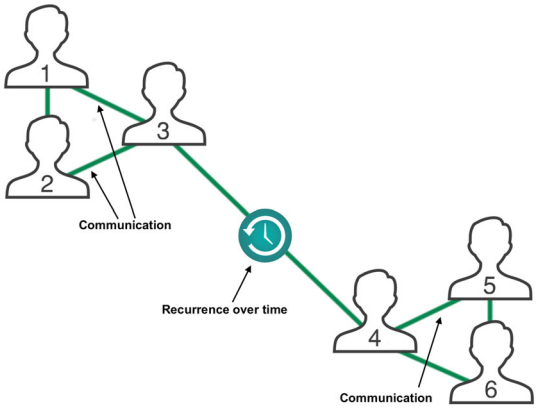

FIGURE 4 Black cloud effect community smell, an identification pattern [Colour figure can be viewed at wileyonlinelibrary.com]

completely exhaustive partition of V_m induced by the clustering algorithm. From the partition, *black cloud* is the set of pairs of developers C that connect otherwise isolated subcommunities.

The smell manifests if the above condition holds for at least two consecutive organizational time-windows (fixed to 3-month intervals, in the case of CODEFACE4SMELLS). An example is presented in Figure 4. Here, the occurrence of *black clouds* reflects two developers, "3" and "4," who are the lone boundary spanners across two different subcommunities and over time—at least two subsequent analysis windows (3 months, in our case).

Detecting black clouds requires eliciting the communication network and applying known community detection algorithms[18] to identify subcommunity structures and boundary spanners across them. For example, see Figure 1 where two subcommunities (previously specified) can be detected by considering the density of communication links.

2.5.4 | Bottleneck effect

The *Bottleneck* community smell is characterized by the occurrence of the following suboptimal characteristics within a software development community: (i) proposed changes within every software development phase require an extraordinary quantity of time to be implemented;[12] (ii) time waste;[13] (iii) hidden or counterintuitive information (and broker) locations;[12] (iv) highly formal or complex organizational structure;[12] (v) highly regularized procedures.[12,13]

The fundamental side-effect generated by this community smell is a massive delay that characterizes key organizational processes within the community such as decision-making, due to personnel unavailability or communication overload. The identification pattern of this smell is based on the detection of unique knowledge and information brokers in different subcommunities.

In our attempt to define an automatic identification pattern for this community smell we focused on the analysis of project communication networks. We considered the six key factors around *Bottleneck* as reflecting the presence, within a project organizational structure, of a unique boundary spanner across several different subcommunities (ie, more than 2). The social-network analysis concept of unique boundary spanner[17] has, in fact, a remarkable similarity to *Bottleneck*. A unique boundary spanner interposes him/herself into every formal interaction across two or more subcommunities and if the organizational structure of the project is complex and characterized by highly formal procedures, it will not be possible to incept parallel information channels between other members of the subcommunities.

To further elaborate on the definition of this community smell, let consider the example proposed in Figure 5. As shown, detecting *Bottleneck* requires the identification of community members who are the *only* members of their subcommunity that communicate with (at least) two other subcommunities. Therefore, assuming a communication link was present between developer "A" and "B," then developer "B" is the pivot of *Bottleneck*.

2.6 | CodeFace4Smells extension

From the perspective of the CODEFACE tool, a *community* is operationalized as a densely connected set of nodes within the community group (ie, the members that make up a development community) which is sparsely connected to all other

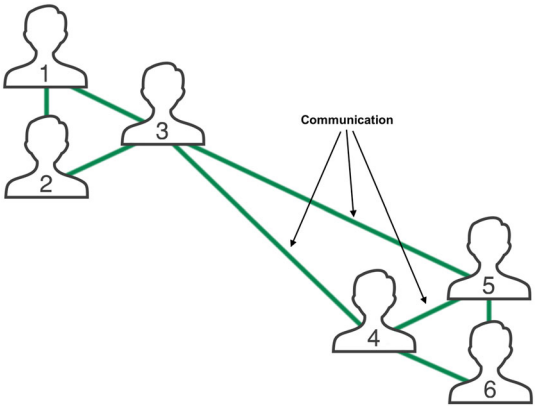

FIGURE 5 Bottleneck community smell [Colour figure can be viewed at wileyonlinelibrary.com]

nodes in the network. To identify and properly characterize the community structure, the CODEFACE tool enacts two community detection strategies, defined as follows, paraphrasing from Joblin et al:[8]

1. *Function*—To recover a community structure, CODEFACE uses a heuristic for identifying when two developers are engaged in a coordinated effort using a fine-grained heuristic based on code structure, where developers are considered to be coordinated when they actually contribute code to a common function block. Furthermore, CODEFACE uses the commits' timestamp for identifying the appropriate directions of the edges in the recovered community structure.
2. *Committer-Author*—In this method, the tool uses tags to identify relationships between all people that contributed to a common commit, including authors, reviewers, and testers. For example, sign-off tags are self-reported acknowledgments of participation on a commit, therefore the tag-based networks undoubtedly capture real-world collaboration.[8]
3. *Community-Verification*—to verify the recovered community structure, CODEFACE uses a random null-model to compute the probability of observing the identified community in an equivalent class of null-model graphs that lack a community structure. The tool generates the null-model using a standard approach called the configuration model for random graphs, where nodes are joined uniformly at random under the constraint that the degree distribution is identical to the observed graph.[19]

To the above heuristics, we add a systematic implementation of the operationalization provided in the previous section, to allow for automated detection of community smells at the same time as CODEFACE operates community structure recovery. The output of the tool is represented by a CSV file containing the community smell instances identified over a social structure representation known as a DSN, a notation previously used for bug prediction and error-proneness.[20,21]

3 | ANALYSIS OF THE RESULTS

3.1 | Perceptions of team members

To address RQ1, we interviewed four key forge managers, designers, developers, and operators who, together, cover the entire 25 year history of the forge. We supplemented this data through the analysis of blogs and news articles that documented the history of SourgeForge. This supplementary data was used to obtain specific details of events described by the participants (eg, to identify the details of a change in ownership). In cases where details were obtained from news articles or blogs, they are referenced.

3.1.1 | Research methods

We performed semistructured interviews. The goal was to understand their perspectives on project and community health. Each interview lasted about 1 hour. The interviews were transcribed by an independent third-party. Thematic content analysis was used to analyze the interview transcripts.[22]

3.1.2 | Results for RQ1

Early Success. We asked the interviewees what factors they perceived to have led to the early success of SOURCEFORGE. The interviewees described three main factors:

- **Emerging Need.** SOURCEFORGE addressed an emerging need. Offering free code hosting when SourgeForge launched in 1999 was revolutionary. One participant said "we wanted like 1000 projects in the first year, and we had 1000 projects in the first month." By 2007, there were more than 150 000 users and over 1.5 million projects.[23] Thus, SOURCEFORGE grew much larger and faster than expected.
- **First To Move.** Similarly, SOURCEFORGE was the first to offer free and versioned code-hosting. As the demand for this service grew, SOURCEFORGE's popularity exploded. This can largely be attributed to the fact that they were the only ones offering this service in the beginning.
- **Skunkwork Team.** Interviewees also suggested that the Skunkworks nature of SOURCEFORGE contributed to its aggressive expansion of popularity. Skunkwork teams are a small group of people who work on a project in an unconventional way; the group's purpose is to develop solutions quickly with minimal management constraints. In the case of SOURCEFORGE, there was no direction from management on what should be developed. The project itself happened by happenstance. The team of four developers were given the goal of generating online traffic. The developers came up with the idea of creating what would eventually become SOURCEFORGE because of their passion for open source software and their belief that it would attract traffic. New features originated within the team. One participant said "we did the development of SOURCEFORGE on SOURCEFORGE. So as we needed a feature ourselves we would write it." Thus, the team had significant freedom in deciding what to develop.

The early success of SOURCEFORGE was not seen to be related to the technical competitiveness of the project, but rather being the first to fill an emerging need.

Eventual Downfall. We also asked the interviewees about the project and team health throughout the project and what (if anything) they would have done differently. We identified several factors that interviewees believe eventually led to the downfall of SOURCEFORGE:

- **Not Considering ROI.** Due to the Skunkwork nature of the project, the team paid little attention to how much money was being spent or how much money was coming in. One participant said "it was never part of the plan to make money." However, after the company went public, interviewees described an emerging need to track the return-on-investment (ROI) that the forge was producing (if any). For example, one of the interviewees mentioned that "[...] a lot of hardware was coming in and nothing was coming out, so people in the high places started to ask questions." Another interviewee said "once we were public and now we had responsibilities to shareholders and things like that, people obviously started wanting to know where all this money was being dumped into, and wanted to know how we were going to return on investment and things like that. And so 6, 8 months later is when we started actually getting pressure from executives to figure out how we were going to make money."
- **Deceit.** Due to the push to increase ROI, in July 2013, SOURCEFORGE introduced a new program, called DevShare. This program bundled third-party software with project downloads, following the model of the widely known CNET download network.[8]

  DevShare was conceived as a way for open source software projects to monetize their efforts while still keeping the software open source and free. The ad revenue would be shared with the projects.[24] The developers thought that enabling ways for the OSS projects to make money was a good goal, but the way DevShare was implemented was too dishonest, putting ROI over trustworthiness. One participant said, "SOURCEFORGE has been trying to help projects to sustain what they were doing with dubious initiative like Devshare. Might have been a good thing if that was run the right way. As a matter of fact today projects still need money and they don't have the solve for that."

  While DevShare was an opt-in service, some projects complained that SOURCEFORGE bundled third-party adware in their downloads without their consent.[25] Ads were added to project download pages with fake download buttons to trick users into clicking on the ad. Often, clicking on these ads resulted in the download of adware. At this time,

[8]http://cnet.com/

many projects announced they were abandoning SOURCEFORGE, citing DevShare as one of the main reasons. The GNU image processor (GIMP) was the first big project to announce it was leaving in November 2013.[25]

- **Two-Masters Syndrome.** Part of the reason DevShare was implemented was driven by some of the bigger OSS projects using SOURCEFORGE at the time. Many large projects called SOURCEFORGE home, including VLC media player and GIMP. At this point, there were plenty of competitors providing source-code hosting, and SOURCEFORGE wanted to keep the projects. One participant said "We had a lot of bigger projects on the site, and then we had lots and lots of little projects right. So the bigger projects, they wanted some kind of revenue sharing. We started catering to the really big projects and trying to implement enough of the little features that the smaller projects wanted to stay as well."

  The two-masters syndrome[26] is a partially unknown organizational effect[27] where the team needs to work to satisfy two "masters" while making money out of both, but both end up having a conflicting agenda that creates an impasse.
- **Organizational instability and disconnect between management/developers.** SOURCEFORGE changed ownership many times. One participant said "every 18 months we'll have a new owner and a new set of managers. At one point in time we all got together for beers and wrote them down, and it was almost every 18 months on the dot." With these changes in ownership, there were also changes in direction and a disconnect between management and the development team. A participant complained that "by 2008 we had already actually changed hands of who owned us and what we had been doing two or three times, that the new people that had come in and acquired us, they didn't care about the open source ethos, they didn't care about anything except for making money back."

  DevShare was introduced after one of the changes of ownership without much consideration for the opinions of the developers, causing many to leave the team. One participant said "Because the company was sold one of the many times it was. But it was sold at that time, so changing hands, the new owners decided to try to give a spin using the Devshare program among other things. So this is not a team decision, it wasn't a team decision, it was of course a company decision. So the idea was, let's try to do this and see if that might help, but if you talk about the developer team, some of them decided to move on and went to join other initiatives. So not everyone decided to stay at SOURCEFORGE at that time."
- **Skeleton Crew.** There were very few core contributors to the SOURCEFORGE project. One interviewee said "there was three main developers and the manager type that was also a developer but working part time. We needed more people, and we just couldn't get people. We needed more hands." The unexpected popularity and growth of SOURCEFORGE coupled with the need to serve two masters with conflicting agendas, meant that the small skeleton crew was insufficient.
- **Blinded by Technical Debt.** The small skeleton crew was further complicated by the grassroots start of the project. At the time when GitHub was released, the SOURCEFORGE team was in the midst of a complete refactoring and redesigning of their code base. One participant said, "A little bit before that, 2008 or whatever, we had started rewriting the entire site, trying to pay down the almost decade of technical debt that we'd accumulated." This effort prevented the team from fully noticing the disruptive change that was occurring in the software landscape with the advent of collaborative development tools.
- **Missed Paradigm Shift.** Meanwhile, new collaborative, highly distributed software development, hosting, and versioning tools were disrupting the market. Interviewees stated they believed these were fads and did not consider modernizing SOURCEFORGE to keep up to date with this paradigm shift. This was likely the result of having an overworked, skeleton crew who did not have time to really step back and look at the changing landscape. One interviewee said, "I don't think we quite got the importance of, you know, of the social element … we were still in very much a sort of dot com 1.0 framework of content producers and content consumers being very distinct populations."

  The quality of service went down and projects started moving away. The exodus of projects was facilitated by the advent of GITHUB, which was launched in 2008. GITHUB, a competing code hosting website, differed from SOURCEFORGE as it was built on top of git, a distributed version control system. GITHUB also offered many collaboration features. In June 2011, ReadWriteWeb reported that GITHUB had surpassed SOURCEFORGE in total number of commits for the period January to May 2011.

There were many factors that seem to have contributed to the downfall of SOURCEFORGE. Yet, all of the factors seem to be connected in various ways. Figure 6 offers a chronicle of the SOURCEFORGE story. The figure plots participation (number of users shown by the continuous line) and size (total lines of code (LoC) added shown by the dotted line) over time. The participation and size numbers were obtained from the SOURCEFORGE research data archive.[23] The figure also highlights the major external events that pertained to SOURCEFORGE (top part) as well as the internal reorganizations (bottom part) where we highlight the reorganization start and finish (forked-line arrow from the bottom). The figure

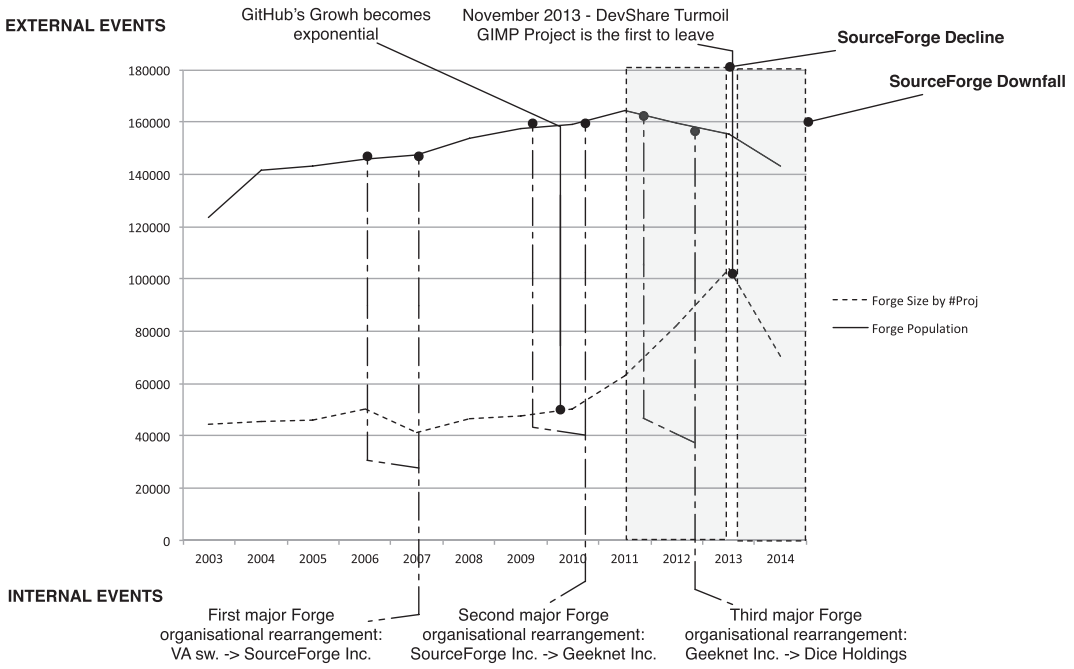

FIGURE 6 A chronicle of SOURCEFORGE using participation (#people, continuous line) and size (total #LoC added, dotted line) over time. Indications of a declining Forge are observable earlier than the DevShare incident and coincided with many developers leaving; downfall (abrupt drops of both people and projects) manifests 2 years later

clearly illustrates a *decline* in the population and size starting in 2011. In 2013, after the introduction of DevShare, we witness the *downfall* of SOURCEFORGE. By downfall, we mean an abrupt, steady, and continuing loss of people and projects.

Summary for RQ1. The results coming from the semistructured interviews confirm that the decline of SOURCEFORGE has been facilitated by both social and technical aspects around the software development community.

3.2 | Community and technical structures of SOURCEFORGE

To address **RQ2**, we studied the evolution of APACHE ALLURA from both social and technical perspectives.

3.2.1 | Finding evidence of social and technical debt

From a sociotechnical perspective, we sought out the most established indicators of social debt,[12,16] namely, (1) sociotechnical incongruences, or breaks in sociotechnical congruence[28] as well as (2) the presence of suboptimal patterns in the community structure, or *community smells*.[29] Specifically, in this study we considered three of the community smells defined by Tamburri et al:[12] organizational silo effect, lone-wolf effect, and bottleneck or "Radio-silence" effect, which was previously defined.

Furthermore, in addition to the previously defined community smells, we considered two well-established measurements for sociotechnical issues, namely:

1. **Smelly Quitters.** This ratio reflects the number of people who were part of a community smell for two subsequent time windows and left the community for the remaining time windows in the available range of data.[30]
2. **Sociotechnical Congruence.** Paraphrased from previous work[31] as *"the state in which a software development organization harbors sufficient coordination capabilities to meet the coordination demands of the technical products under*

development" and operationalized in this study as the number of development collaborations that do communicate over the total number of collaboration links present in the collaboration network.

Along with community-related information, we also collected technical data. In the first instance, we studied how the quantity and sizes of file changes applied by developers vary during the evolution of ALLURA. Second, we extracted data related to technical debt. In particular, we considered two different sources of information:

1. **Code smells.** These represent suboptimal design or implementation solutions applied by contributors during the evolution of the project.[32] While the most-known code smell detectors only work for Java programs,[33] we needed an automatic tool able to identify design issues in Python. For this reason, we employed the detector proposed by Chen et al,[34] which can identify 10 different smell types—including both traditional code smells (eg, Large Class and Long Method[32]) and Python-specific ones (eg, Long Ternary Conditional Expression[34]). We relied on the original implementation of the tool made available by the authors. It is important to note that this detector is the only one currently supporting the detection of Python smells; its accuracy has been reported to be high (average precision of 98%), being therefore suitable for our purpose.
2. **Self-admitted technical debt (SATD).** Potdar and Shihab[35] defined SATD as a practice that developers use to admit the existence of temporary design solutions (at any level, from requirement to code debt) that should be fixed. To identify SATD, we exploited regular expressions to match inside comments the 62 patterns defined by Potdar and Shihab.[35] This list includes a set of keywords that are likely to indicate the presence of a SATD (eg, *fixme*, *this is a hack*, and so on), and has been defined by manually analyzing more than 100 000 code comments.[35]

3.2.2 | Results for RQ2

Results of our sociotechnical data synthesis and analyses are reported in Figures 7 to 10. In the following, we first report on the evolution of number of developers as well as community smells in the considered period; furthermore, we describe how additional sociotechnical factors such as the sociotechnical congruence and the smelly quitter ratio evolved in the same period.

First, Figure 7 outlines the numbers pertaining to core (ie, contributing to both communication and committing) and periphery contributors (ie, contributing to communications only) to the APACHE ALLURA project. The figure shows a heavy fluctuation of all contributor types across our data sample, with a standard deviation of 4.6 developers per every 3-month snapshot, meaning that the community acquired and lost an average of four to five developers around every considered snapshot.

Furthermore, Figure 8 outlines the progression of the reported numbers of community smells in project APACHE ALLURA—the figure also highlights in the gray area to the left-hand side, the period we denoted with downfall on Figure 6.

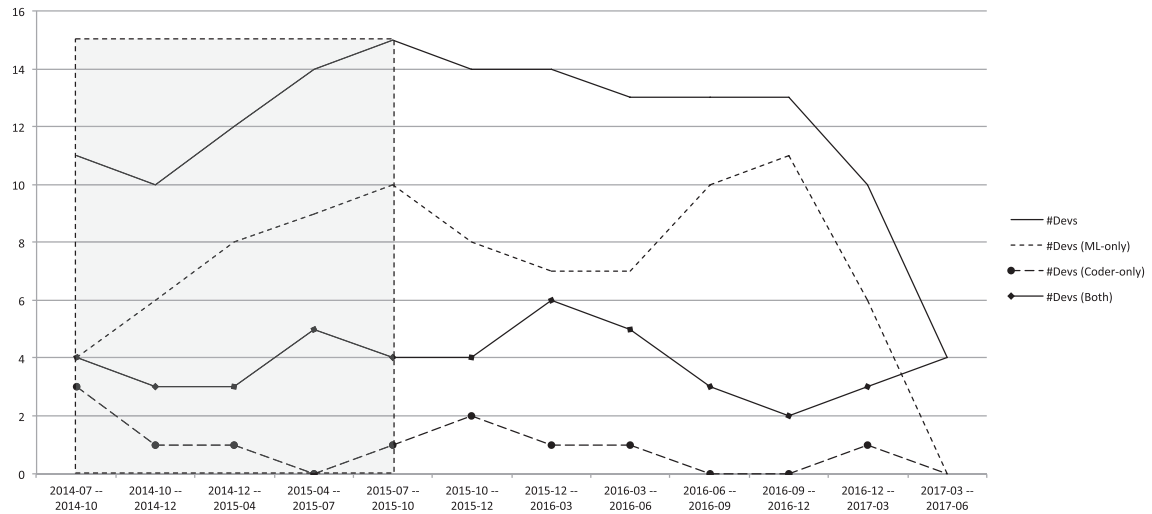

FIGURE 7 Numbers of developers for APACHE ALLURA—highlighted is the period we denoted with downfall

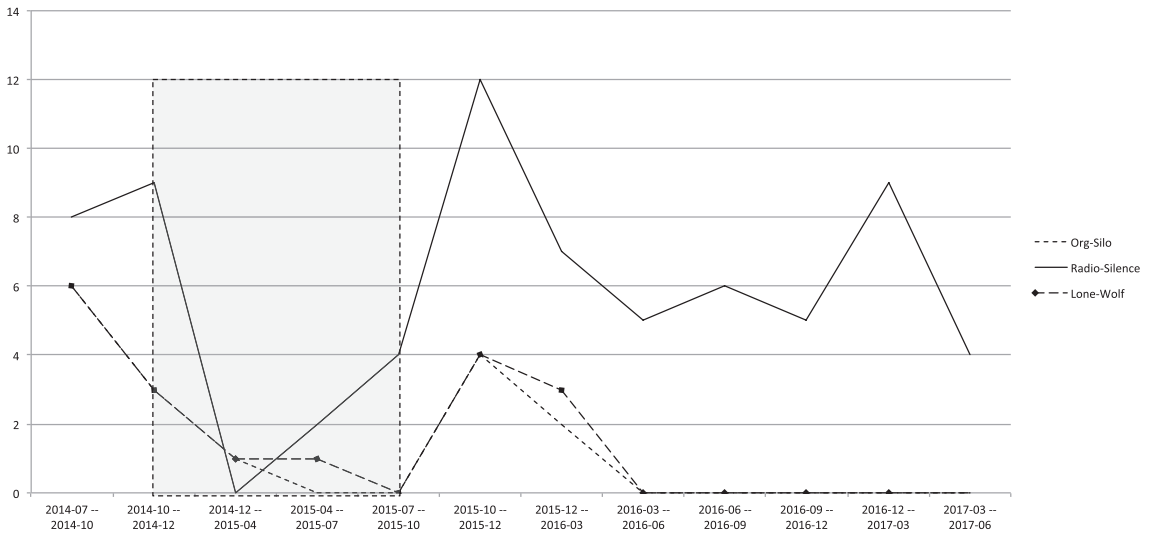

FIGURE 8 Numbers of community smells for APACHE ALLURA—highlighted is the period we denoted with downfall

The figure highlights a coincidental drop of community smells (deviation of −60%) all along the entire period of downfall. The growth of smells begins again between late 2014 and mid-2015, precisely around the corporate acquisition of SOURCEFORGE/ALLURA by yet another external company, the well-known videogame giant GameStop, in this case. The figure also shows an almost identical drop (standard deviation is 0,5) of the number of community smells coincidentally to a final acquisition by SOURCEFORGE/ALLURA between the end of 2015 and mid 2016 by its current owner, private company BIZX.

Finally, Figure 10 plots results for the smelly quitter ratio across APACHE ALLURA. The figure shows a recurrent pattern clearly appearing twice (see the grayed-out areas on the figure, degree of isomorphism >80%) in the project's community structure and, not surprisingly, once again coinciding with the two subsequent organizational rewirings in the respective periods.

Summary for RQ2, Part (a)—Socio-Organizational Perspective. Our evidence shows recurrent organizational turmoil patterns that coincide with organizational rewiring scenarios around the forge. This evidence suggests a lack of organizational stability in SOURCEFORGE/ALLURA.

Furthermore, while organizational change is inevitable, most organizations do not track it. Tracking emerging metrics such as the number of community smells or the ratio of "smelly" quitters offers a reliable and fine-grained (eg, see Figure 10) perspective over the effects and manifestations of organizational rewiring.

The recurrence of a drop in smelly quitters likely indicates a recurring turmoil coinciding with the organizational rewiring taking place around the forge. The recurrence of the above patterns is reflected partly on sociotechnical congruence (see Figure 9) and even more clearly in the smelly quitters ratio (see Figure 10). In particular, Figure 9 shows steadily increasing congruence values throughout 2014 and late 2015, when a blatant drop in sociotechnical congruence happens around APACHE ALLURA, at the time of the acquisition by BIZX. This suggests a misalignment between the organizational structure of developers and operators with respect to the acquisition by BIZX; the drop likely indicates the turnover surrounding this acquisition.

From a technical perspective, we analyzed the (i) quantity and sizes for code changes applied and (ii) code smells and SATD detected on APACHE ALLURA over the entire analysis time-window running from early 2014 until late 2016, extracting essential code quality metrics to control code quality as well. Figures 11 to 14outline the achieved findings.

In particular, we observed that the sociotechnical observations made in Sec. 3.2.2 recur. For example, in coincidence to the same organizational rewiring scenarios, code-changes are localized (eg, the top-right plot from Figure 11 highlights an series of changes focused on five to seven files), the difference-in-size range is almost constant (bottom-left plot on Figure 11, range fluctuates around 100 lines) and the code-commit tags disappear after the first two periods of our analysis while the size of commit messages is almost constant and quite considerable, between 50 and 100 words—this evidence seems to denote a constantly changing code-base around an organization that disconcerts the use of typical Apache projects' coordination practices such as commit-tagging.

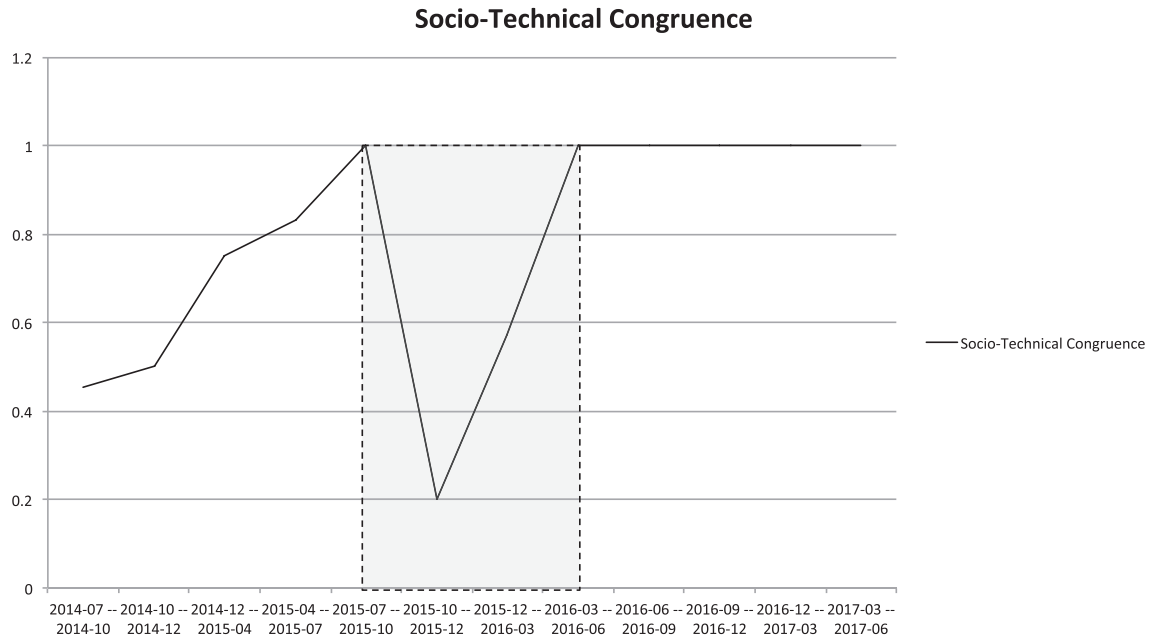

FIGURE 9 Sociotechnical congruence for APACHE ALLURA—highlighted is the period we denoted with downfall

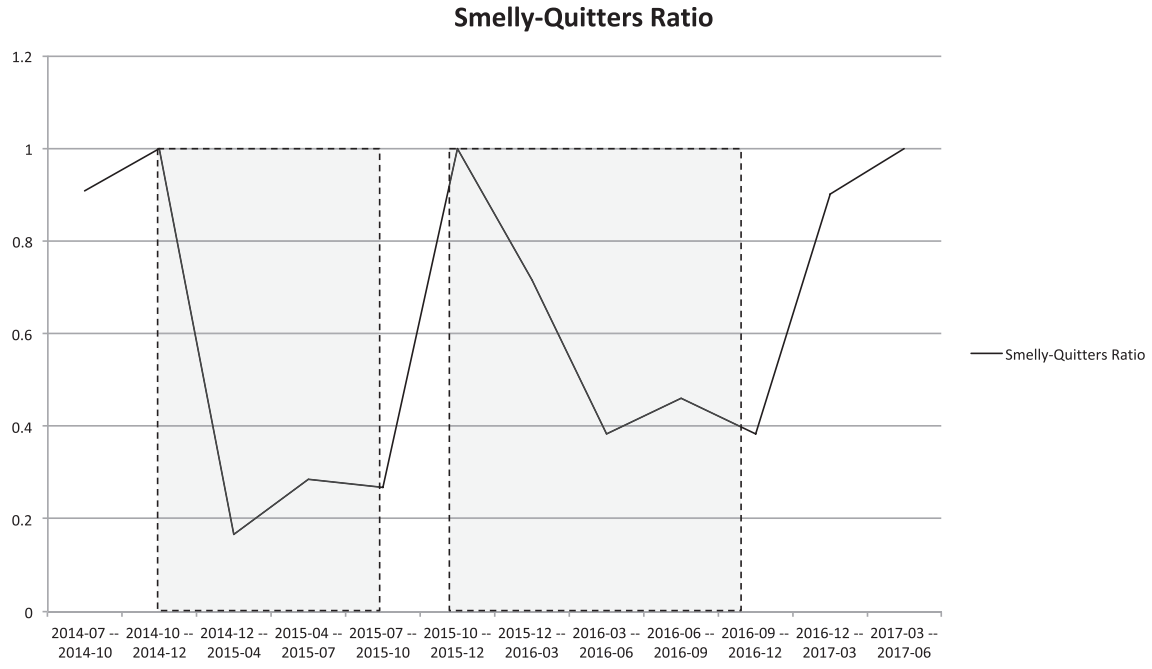

FIGURE 10 Smelly quitter ratio for APACHE ALLURA—highlighted is the period we denoted with downfall

Similarly, Figure 12 provides another confirmation of our organizational turmoil observations as reflected on the technical artifacts. In particular, all highest plot "peaks" in the top-most and second plot on Figure 12, reflect time-ranges coinciding with the aforementioned organizational turmoil. Even more importantly, in coincidence with the last organizational acquisition, a massive campaign of code-changes was initiated, as if the forge system needed again to undergo major redesign.

Furthermore, Figures 13 and 14 are perfectly in line with the results discussed so far. Indeed, we observe that during the time period referred as downfall in Figure 1 both code smells and SATD tended to decrease, while their numbers increase once the organizational structure of ALLURA became unstable. It is important to remark that, despite the

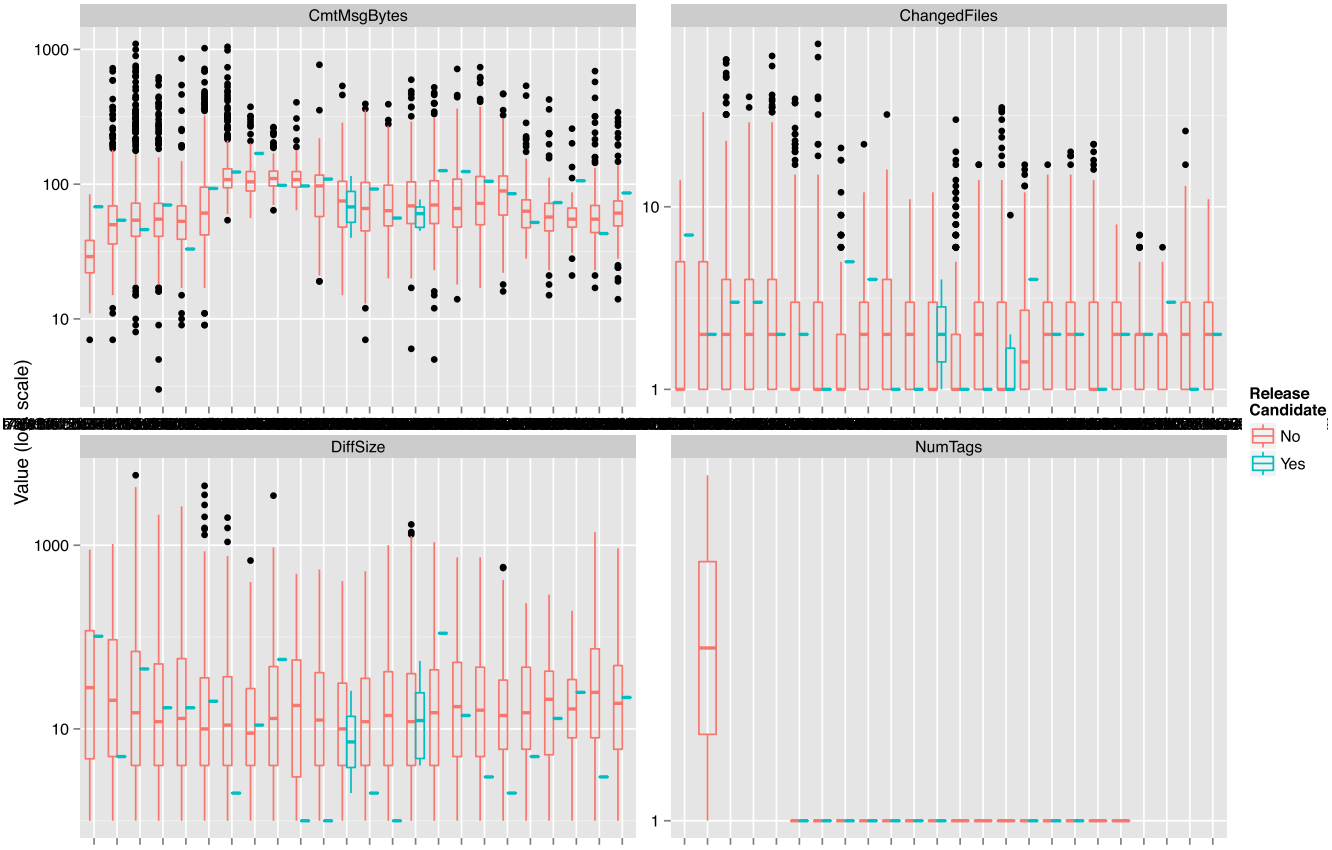

FIGURE 11 APACHE ALLURA change sizes (Y-axis) per commit (X-axis) [Colour figure can be viewed at wileyonlinelibrary.com]

absolute number of code smells and technical debt is not that high (eg, at most, we observed 23 Large Class instances), it may still substantially contribute to the drift of the design of a software system, possibly compromising the overall quality—as demonstrated by previous work in the field.[36,37] The observations above are especially true when considering the joint fluctuation between community smells and three particular code smell types, that is, Large Class, Long Method, and Long Parameter List. This close relation is confirmed from a statistical point of view: specifically, we applied the Granger causality test[38] to determine whether one time series (ie, the introduction/removal of community smells) is useful in forecasting another (ie, the introduction/removal of code smells). In other words, we tested whether the presence of a community smell c_i can be used to "predict" the presence of a code smell cs_i. Note that we used the Granger test instead of association rule discovery[39] because we are interested in assessing the statistical significance of the temporal relation between community and code smells, rather than of just their cooccurrences. As a result, we found that relation between community smells and the large class, long method, and long parameter list is significant (all the ρ−values are <0.001).

Summary for RQ2, Part (b)—Technical Perspective. Our evidence shows recurrent organizational turmoil patterns influencing technical stability. The combined technical and socio-organizational evidence denotes an organizational and technical structure in downfall, featuring major restructurings and negative manifestations across both perspectives.

3.3 | Architectural decisions of SOURCEFORGE

To address **RQ3** we analyzed the software architecture of eight releases of the Apache Allura project covering a 4 year time-span, from early 2014 to early 2018. These were releases: 1.1.0, 1.2.0, 1.3.0, 1.4.0, 1.5.0, 1.6.0, 1.7.0, and 1.8.1.

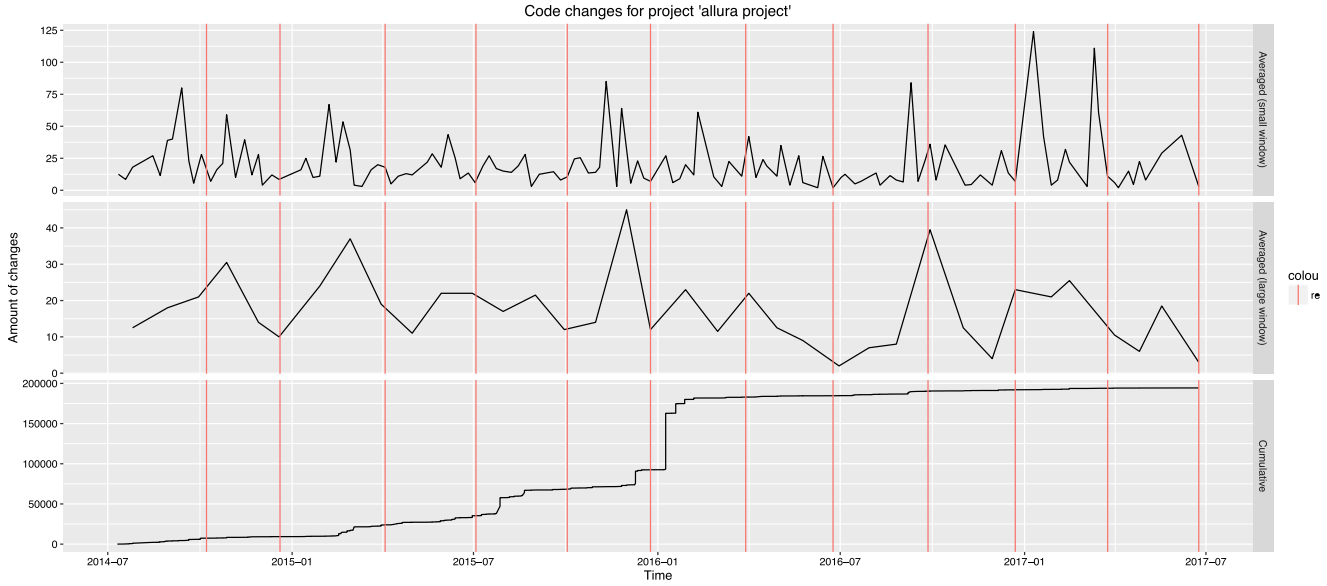

FIGURE 12 APACHE ALLURA change quantity over time [Colour figure can be viewed at wileyonlinelibrary.com]

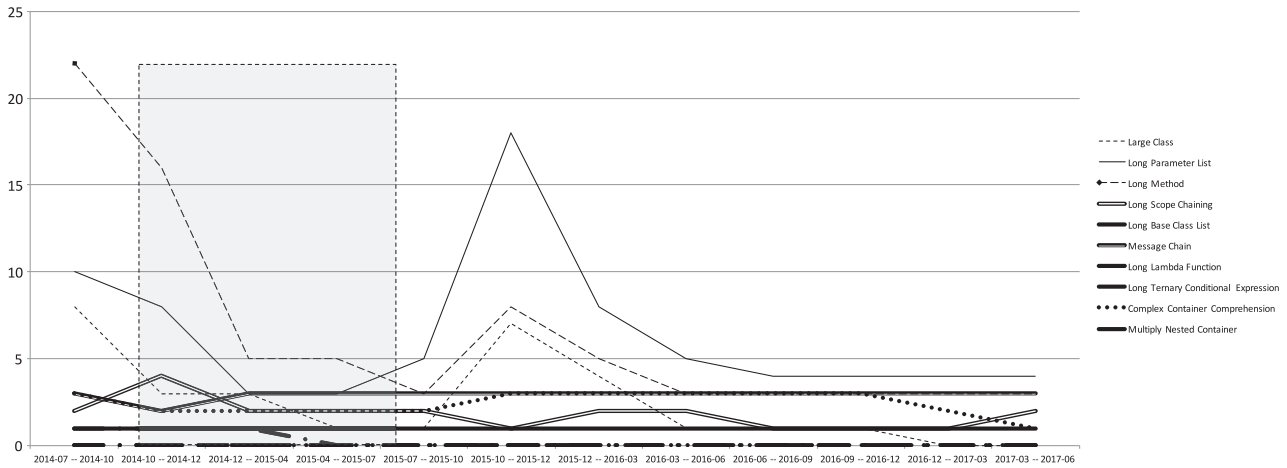

FIGURE 13 Numbers of code smells for APACHE ALLURA; an evident peak reflects the frantic refactoring connected to the period in which the decline started, a clear indication of technical turmoil and haphazard technical debt generation—highlighted is the period we denoted with downfall

3.3.1 | Research methods

We analyzed the software architecture of each of these releases. Or, more precisely, we analyzed the module structure of each of these releases using a commercial version of the Titan tool suite[40] called DV8.[9]This tool suite allowed us to reverse engineer each release of Allura, and perform two analyses:

- measure the DL of the module structure of each release;[41]
- measure the architectural flaws in the module and package structure of each release;[41]

Each of these measures has been shown to be highly correlated with maintenance effort and bugginess.[41]

[9]https://www.archdia.net/

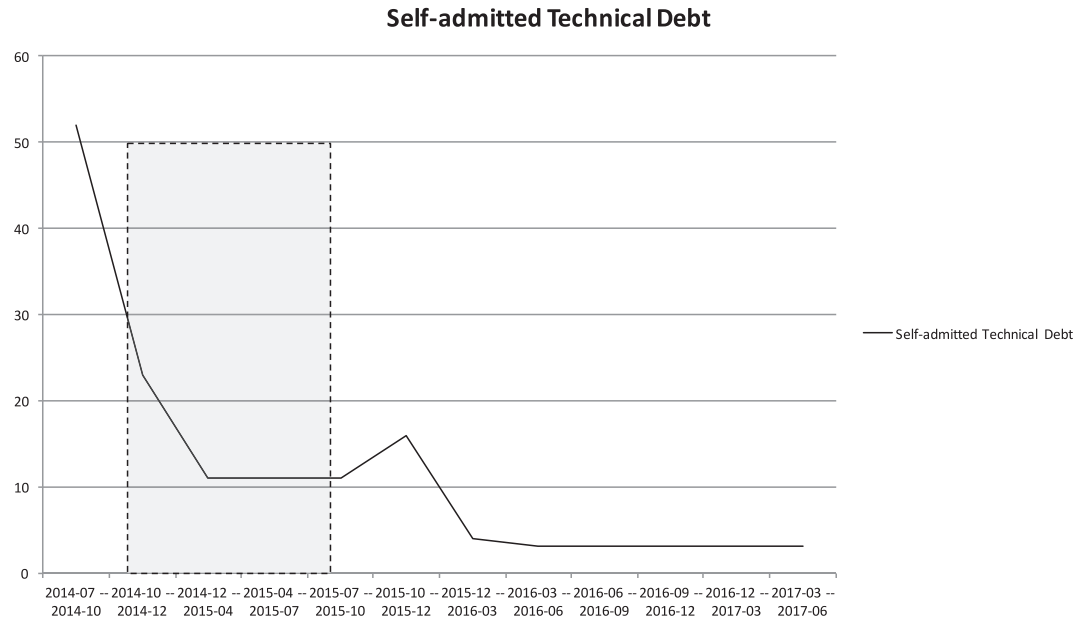

FIGURE 14 Numbers of SATD items for APACHE ALLURA; the conflict between the SATD and the smells reported indicates an extent of failure reticence, people cannot admit the frantic refactoring is not helping but only making things worse—highlighted is the period we denoted with downfall. SATD, self-admitted technical debt

3.3.2 | Results for RQ3

The DL scores of all eight releases are shown in Table 1. There are two points to note about these scores:

- The scores are uniformly low. These DL scores place APACHE ALLURA in the bottom 10th percentile of all projects analyzed in Reference 41, where 129 projects, both commercial and open source, were measured.
- The DL scores are relatively stable over time, decreasing slightly from release 1.2.0 to 1.8.1. This suggests that the team was not attempting to address any of the architecture debt[42] that had accumulated.

DL is a measure of how (de)coupled the system's source code files are. A low measure suggests a highly coupled system, one that is likely to be difficult to understand, debug, and modify. But this is just a single number. To understand these low DL scores more precisely we analyzed the architectural flaws (also known as "hotspots") in each of the APACHE ALLURA releases. These flaws have been shown, in several studies,[41,43] to be very strongly correlated with bugs, changes, and churn across a wide variety of projects.

TABLE 1 DL scores from eight releases of APACHE ALLURA

Release	Date	DL score
1.1.0	1/2014	32%
1.2.0	12/2014	34.6%
1.3.0	6/2015	34.6%
1.4.0	4/2016	33.8%
1.5.0	8/2016	33.7%
1.6.0	12/2016	33.5%
1.7.0	3/2017	33.5%
1.8.1	3/2018	33.5%

Abbreviation: DL, decoupling level.

Release	UnhInt	Clique	ModVio	PkgCyc
1.1.0	11	5	186	17
1.2.0	13	6	232	17
1.3.0	13	6	238	18
1.4.0	13	4	254	17
1.5.0	13	4	254	18
1.6.0	13	4	256	18
1.7.0	13	4	253	17
1.8.1	13	4	250	18

TABLE 2 Architectural flaws in eight releases of APACHE ALLURA

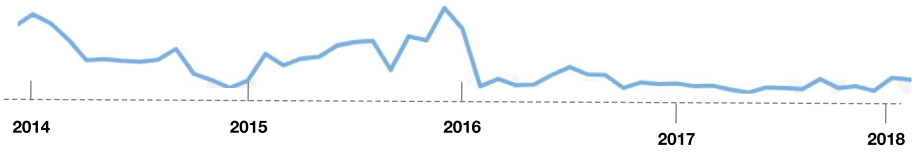

FIGURE 15 APACHE ALLURA activity, in commits per month, 2014 to 2018—picture was extracted via OpenHub [Colour figure can be viewed at wileyonlinelibrary.com]

The rationale behind performing this analysis is that we wanted to measure the flaws over time. This would help us understand whether the developers were attempting to pay down the architecture debt, by removing at least some of the architectural flaws.

As for the DL analysis, the results are summarized in Table 2. In this table, we report on the number of instances of four flaw types: Unhealthy Inheritance (UnhInt), Cliques (Clique), Modularity Violations (ModVio), and Package Cycles (PkgCyc). These flaw types are described in Reference 41. As can be seen from the table, number of flaws in this project is large and it remains stable or even increases slightly from release to release. This suggests that the architecture had a heavy load of debt and that this debt was not being addressed. As such it was increasing slightly with each subsequent release, making the project harder to maintain and evolve.

This observation is consistent with the recorded activity, in terms of commits, on the Allura project over the studied time-frame. Over this time-frame commit activity dropped precipitously, as shown in Figure 15, taken from openhub.com. In January 2014 there were 140 commits per month. By January 2018 there were just 28 commits per month. While we cannot claim a causal relation here, this change in activity is consistent with our expectations.

Summary for RQ3. The project had a heavy and continuously increasing load of architectural debt and this debt was never repaid.

4 | DISCUSSION AND IMPLICATIONS

We identified a variety of both social and factors that contributed to the decline and downfall of SOURCEFORGE. These factors were both internal and external. SOURCEFORGE's early success was because they were the first to fill a need. At the time, their skunkwork team was a benefit and enabled them to identify and fill a gap in the market quickly. However, as the popularity of SOURCEFORGE grew, the skunkworks team was no longer advantageous. The team tried to serve two masters (small projects and large projects) with different and, at times, conflicting needs. The skeleton crew introduced more and more technical debt as they tried to meet the needs of all of their stakeholders quickly. While they focused on dealing with technical debt and continued to try to satisfy the needs of their stakeholders, they missed a large paradigm shift that was occurring—the movement to distributed version control systems and more collaborative development tools. GitHub's growth became exponential while the SOURCEFORGE team largely dismissed it as a fad.

Meanwhile, SOURCEFORGE changed ownership several times, and a disconnect between the development and management started to cause major problems. Management focused on ROI; SOURCEFORGE was expensive to run and did not have a plan to bring in revenue. This led to the introduction of DevShare, but since management did not understand the open source ethos and the development team was not included in management decisions, DevShare was a major failure. It prioritized ROI over trust and bundled adware with project downloads. Many projects started leaving SOURCEFORGE, citing DevShare as a main reason.

Through analysis of the social and technical structures of the project, we also observed a lack of organizational stability and technical and architectural debt. There is substantial indication of community smells presence in the organizational structure around SOURCEFORGE, indicating intense organizational turmoil and lack of stability;[44] at the same time the architecture of Allura is reportedly highly coupled, replete with flaws. The team around the forge never attempted to pay down this architecture debt, presumably too focused on paying back the more basic, code-level, and SATD.

There are several lessons that can be learned from this cautionary tale. Here, we highlight lesson for other software projects and directions for future research.

Advice: Determine business plan early. One of the key factors that contributed to the major exodus of projects was DevShare. While the deceitful nature of its implementation is largely to blame, it also illustrates how important it is for a software project to identify its ROI strategy early. DevShare was an effort to increase ROI since the original project did not have a plan.

Advice: Management and development teams must be aligned. Another contributing factor to the downfall was the misalignment between management and the development team. The developers clearly indicated that they were not involved in key decisions, despite having more expertise in the domain. This disconnect caused an already small team to lose valuable talent, but also caused poor decisions that did not align with the ethos of the community their product served. The aforementioned condition is a known phenomenon in organizations and social networks research often referred to as "two-masters syndrome",[45,46] which could itself be a *community smell* which was not previously manifested nor formalized in software engineering research and practice and may deserve further attention at least to support collaborative and computer supported cooperative work in the scope of software production and operation platforms such as GitHub.

Advice: Be careful of changing paradigms. There were many reasons for the missed paradigm shift, particularly they had an overworked, skeleton crew who were blinded by technical debt. Yet, one important aspect to discuss is the expertise of the team. The developers of SOURCEFORGE were designing and developing software for a domain in their own area of expertise—software development. They were very comfortable with centralized version control systems and did not appreciate the importance of incorporating collaboration into software development tools. This high level of domain expertise, in a way, hindered their ability to appreciate the change that was happening since they were very likely considering their own needs as a software developer. Given the small size of their own team, collaborative tools may not have seemed important.

Software projects should be careful to continue to assess the landscape in which they operate. Diverse teams can help ensure different perspectives about changing landscapes are considered.

Future research: More comprehensive project and community health metrics are needed. Our evidence showed that SOURCEFORGE suffered from a variety of social and technical problems. We cannot claim that additional project and community health tracking would have prevented the downfall of SOURCEFORGE, but we do observe that using some of the community smells and technical and architectural debt metrics we examined in this study would have revealed problems. Potentially if some of the problems had been revealed earlier, the fate of SOURCEFORGE would have been different. Future research should develop tools that enable the tracking of more fine-grained metrics related to the community, the code, and the architecture.

Turnover (eg, as measured by its earlier manifestations such as the Bus-Factor[47]) and smelly quitters are the only two quantities that we investigated which exhibit variability both at the microstructure (single or small-world network interactions) and at the macro-structure level. The only apparent exception to the turnover and smelly quitters variability in the organizational structure of the forge under study is in the very beginning of our sample of observations, namely, the period between early 2014 and mid-2015, where the deviation for both drops by half a point—this period coincides almost identically to the *downfall* time range from the timeline reported in Figure 6. The coincidence could indicate that Forge designers and managers detected the negative trend, trying to pick up the disaster scenario caused by DevShare and sought external collaboration as well as increasing the number of paid maintainers. On the other hand, the indication in question could have been aided by automated-tracking of turnover and smelly quitters[11] emerging from organizational tracking. *Further research should be invested in these factors to correctly establish the feasibility of the aforementioned metrics in this context.*

There is also an intrinsic relationship between the social, technical, and architectural factors. Thus, additional comprehensive metrics should be developed beyond the existing measures like sociotechnical congruence. While some initial work on this has been carried out,[11] *more research is needed on the relation between social and technical debt as well as on the methods for assisting developers in reducing the joint issues coming from such relation.* Some communities are already forming to develop additional health metrics, for example, the CHAOSS initiative[10] or the SECOhealth FNRS international project[11]. Future efforts should develop tools that enable comprehensive analysis of project and community health that consider the relationships between the social, technical, and architectural components of a project.

Future research: Analysis of failures and Retrospectives. While there has been some recent work that studies the failure of software projects (described in Section 6), additional investigations are needed to understand more deeply the reasons for failure of declines of software projects. Additional case studies could be performed to add to the evidence collected here.

5 | THREATS TO VALIDITY

Like any study of comparable magnitude and scale, this study is affected by several threats to validity. In what follows we outline the major ones in our study design and execution.

Internal and Sampling Validity. Internal validity refers to the internal consistency and structural integrity of the empirical research design. We focused our study on APACHE ALLURA and SOURCEFORGE individually, operating a mixed-methods research approach also adopting several observer, data, and sample triangulation strategies (eg, an additional set of our study of the Forge features an interview dataset whose coding was executed twice and K_α evaluated) being adopted. This notwithstanding, there are up to 90 factors from the state-of-the-art in organizations research[48] that may still be affecting our findings and results. In addition, the quantities and effect sizes of the factors themselves were not addressed in this study. Stemming from this limitation, we are planning further study of our target subjects in follow-up quantitative and qualitative research over the projects hosted on SOURCEFORGE or further interested parties involved with the forge (eg, developers for those projects) that may confirm the validity of this work.

External Validity. External validity is the degree to which results from a study may be generalized to other contexts. One threat to external validity is the small size of our interviewee pool—just four project members were interviewed. While four is a small number of interviewees, they represent a large percentage of the core developer team, and so we believe that their opinions are likely to be accurate representations of the project. Furthermore, our analysis considered only one organization, SOURCEFORGE, so may not generalize to other projects. Future studies should investigate if the social and technical factors we identified also contribute to the downfall of other software projects.

Conclusion Validity. Conclusion validity represents the degree to which conclusions about the relationship among variables are reasonable. In the scope of the discussions of our results we made sure to minimize possible interpretations, designing the study with reference to known hypotheses. In addition, our conclusions were drawn from statistical analysis of our dataset and analysis of the source code of the eight ALLURA releases we could get grips onto. Further data triangulation might improve conclusion validity.

6 | RELATED WORK

While the majority of research on OSS projects has focused on their successes, a number of articles have investigated reasons for their failure. For example, Coelho and Valente[49] report on the results of a survey of 104 developers of failed (deprecated) GITHUB projects. As a result of this survey they provide 9 reasons for failure, such as: "usurped by competitor" (27 projects), "project is obsolete" (20 projects), or "lack of time of the main contributor" (18 projects). Khondhu et al[50] analyzed a set of SOURCEFORGE projects and classified them into "active," "dormant," and "inactive." They then analyzed the "maintainability index" (MI)[51] of the code form a sample of each of those sets of projects. Their results, however, were inconclusive; there was not clear trend in the MI between the different sets of projects, and the sample sizes were not large. Lee et al[52] conducted a study where they identified "five determinants of OSS success and the relationships among them through a literature review of previous IS success models." These factors were: Software quality, OSS

[10] https://chaoss.community/
[11] https://secohealth.github.io/

use, Community service quality, User satisfaction, and Individual net benefits. Their key finding was that "usage of OSS is predominantly determined by user satisfaction and software quality."

There has also been a substantial literature on software project failures: while this research did not specifically focus on OSS projects, a complete overview is available in Reference 53. A number of previous works have investigated factors likely to influence the success/failures of software projects. For instance, Capiluppi et al[54] investigated four dimensions, that is, (i) community of developers, (ii) community of users, (iii) modularity and documentation, and (iv) software evolution, of 406 projects coming from a deprecated open source repository called FreshMeat. They found that most of the projects (57%) have one or two developers and that only a few of them (15%) can be considered active. Tourani et al[55] investigate the impact of codes of conduct in open source projects, finding that they aim at providing a safe and inclusive community to avoid community-related issues. Ye and Kishida[56] conducted a study on the motivations leading developers to engage open source development, finding that learning is the major reason that motivates people to start contributing to open source projects. However, when this need is not satisfied, contributors tend to quit the project, possibly creating critical issues for its success. As a matter of fact, Avelino et al[57] found that the survival of popular GitHub projects heavily depend on one or two developers.

7 | CONCLUSIONS

This article tells the cautionary tale of the downfall of Allura/SourceForge. We identified both internal and external causes, both social and technical in nature. We also found evidence of sociotechnical and architectural problems in project archives. Project and community health metrics could have been used to predict and understand the downfall by tracking its early manifestations. The team reported to be aware of the technical debt in the project, yet this was insufficient to maintain the health of the project. Community managers should also measure and manage the mutual impacts of technical and social debt. Furthermore, our analysis suggests that a combination of managerial and technical flaws doomed this project: architectural flaws, power-distance, and measurable management mishaps eventually led to the forge's demise. Practitioners and maintainers of other software projects can benefit from this cautionary tale, for example, as indications of **what not to do**, in the scope of their community and architecture management infrastructure.

ACKNOWLEDGEMENTS

This research is partially supported by the European Commission grant no. 825480 (H2020), SODALITE, and by the grant no. 825040 (H2020), RADON; finally, by grant ANPCyT PICT-1725-2017. Palomba gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Project No. PZ00P2_186090 (TED).

ORCID

Damian Andrew Tamburri https://orcid.org/0000-0003-1230-8961

REFERENCES

1. Fitzgerald B, Feller J. A further investigation of open source software: community, co-ordination, code quality and security issues. *Inf Syst J*. 2002;12(1):3-6.
2. Mens T, Claes M, Grosjean P. Ecos: ecological studies of open source software ecosystems. Paper presented at: Proceedings of the 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE); 2014:403-406; IEEE.
3. Liu Y, Stroulia E, Erdogmus H. Understanding the open-source software development process: a case study with CVSChecker. Paper presented at: Proceedings of the Intlernational Conference on Open Source Systems; 2005:154-161.
4. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering: An Introduction*. Norwell, MA: Kluwer Academic Publishers; 2000.
5. Xu J, Christley S, Madey G. Application of social network analysis to the study of open source software. *The Economics of Open Source Software Development*. Amsterdam, Netherlands: Elsevier B.V.; 2006:247-269.
6. David PA, Rullani F. Dynamics of innovation in an open source collaboration environment: lurking, laboring, and launching floss projects on sourceforge. *Ind Corporat Change*. 2008;17(4):647-710.
7. The Apache Software Foundation Blog The apache software foundation announces apache allura as a top-level project; 2014. https://tinyurl.com/qjq848y.
8. Joblin M, Mauerer W, Apel S, Siegmund J, Riehle D. From developer networks to verified communities: A fine-grained approach. In: Bertolino A, Canfora G, Elbaum SG, eds. *ICSE*. Vol 1. Washington, D.C.: IEEE Computer Society; 2015:563-573.

9. Hogan B, Carrasco JA, Wellman B. Visualizing personal networks: working with participant-aided sociograms. *Field Methods*. 2007;19(2):116-144.
10. Tamburri DAA, Palomba F, Kazman R. Exploring community smells in open-source: an automated approach. *IEEE Trans Softw Eng*. 2019;1–1. in press
11. Palomba F, Tamburri DA, Fontana FA, Oliveto R, Zaidman A, Serebrenik A. Beyond technical aspects: how do community smells influence the intensity of code smells. *IEEE Trans Softw Eng*. 2018.
12. Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. Social debt in software engineering: insights from industry. *J Internet Serv Appl*, 6(1):10:1–10:17, 2015.
13. Damian A. Tamburri, Patricia Lago, and Hans van Vliet. Uncovering latent social communities in software development. *IEEE Softw*, 30(1):29 –36, jan.-feb. 2013.
14. Huang Q, Liu H, Zhong X. The impact of transactive memory systems on team performance. *IT People*. 2013;26(2):191-212.
15. Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. Smells like teen spirit: Improving bug prediction performance using the intensity of code smells. In*ICSME*, pages 244–255. Washington D.C.: IEEE Computer Society, 2016.
16. Tamburri DA, Di Nitto E. When software architecture leads to social debt. In: Bass L, Lago P, Kruchten P, eds. *WICSA*. Washington D.C.: IEEE Computer Society; 2015:61-64.
17. Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications. Number 8 in Structural Analysis in the Social Sciences*. 1st. Cambridge, MA: Cambridge University Press; 1994.
18. Newman MEJ. Fast algorithm for detecting community structure in networks. *Phys Rev E*. 2003;69:66–133.
19. Gkantsidis C, Mihail M, Zegura EW. The markov chain simulation method for generating connected power law random graphs. In: Ladner RE, ed. *ALENEX*. Washington D.C.: SIAM; 2003:16-25.
20. Pinzger M, Nagappan N, Murphy B. Can developer social networks predict failures? Paper presented at: Proceedings of the 16th ACM Sigsoft International Symposium on Foundations of Software Engineering FSE '08; 2008.
21. Bird C, Nagappan N, Gall HC, Murphy B, Devanbu PT. Putting it all together: using socio-technical networks to predict failures. Paper presented at: Proceedings of the ISSRE; 2009:109-119; IEEE Computer Society.
22. Braun V, Clarke V. Using thematic analysis in psychology. *Qualitat Res Psychol*. 2006;3(2):77-101.
23. Van Antwerp M, Madey G. Advances in the sourceforge research data archive. Paper presented at: Proceedings of the Workshop on Public Data about Software Development (WoPDaSD) at the 4th International Conference on Open Source Systems; 2008; Milan, Italy.
24. Galoppini R. Today we offer devshare (beta), a sustainable way to fund open source software; 2013. https://sourceforge.net/blog/today-we-offer-devshare-beta-a-sustainable-way-to-fund-open-source-software/.
25. Prokoudine A. Anatomy of sourceforge/gimp controversy; 2015. http://libregraphicsworld.org/blog/entry/anatomy-of-sourceforge-gimp-controversy.
26. Hoos F, Messier W, Smith J, Tandy P. The effects of serving two masters and using the internal audit function as a management training ground on internal auditors' objectivity; 2014.
27. Crandall D, Cosley D, Huttenlocher D, Kleinberg J, Suri S. Feedback effects between similarity and social influence in online communities. Paper presented at: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08; 2008; 160-168, New York, NY, ACM.
28. Cataldo M, Herbsleb JD, Carley KM. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In: Rombach HD, Elbaum SG, Münch J, eds. *ESEM*. New York, NY: ACM; 2008:2-11.
29. Tamburri DA, Kazman R, Fahimi H. The architect's role in community shepherding. *IEEE Softw*. 2016;33(6):70-79.
30. Magnoni S. An approach to measure community smells in software development communities; 2016.
31. Valetto G, Helander M, Ehrlich K, Chulani S, Wegman M, Williams C. Using software repositories to investigate socio-technical congruence in development projects. Paper presented at: Proceedings of the International Workshop on, Mining Software Repositories 0:25; 2007; IEEE Computer Society, Los Alamitos, CA. https://doi.org/10.1109/MSR.2007.33.
32. Fowler M. *Refactoring: Improving the Design of Existing Code*. Boston, MA: Addison-Wesley Longman Publishing Co.Inc; 1999.
33. Fernandes E, Oliveira J, Vale G, Paiva T, Figueiredo E. A review-based comparative study of bad smell detection tools. Paper presented at: Proceeding of the International Conference on Evaluation and Assessment in Software Engineering, EASE '16; 2016:18:1-18:12; ACM, New York, NY.
34. Chen Z, Chen L, Ma W, Xu B. Detecting code smells in python programs: Paper presented at: Proceeding of the Software Analysis, Testing and Evolution (SATE), International Conference; 2016:18-23; IEEE.
35. Potdar A, Shihab E. An exploratory study on self-admitted technical debt. Paper presented at: Proceeding of the Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference; 2014:91-100; IEEE.
36. Chatzigeorgiou A, Manakos A. Investigating the evolution of bad smells in object-oriented code. Paper presented at: Proceeding of the 2010 7th International Conference on the Quality of Information and Communications Technology; 2010:106-115; IEEE.
37. Palomba F, Bavota G, Di Penta M, Fasano F, Oliveto R, De Lucia A. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Emp Softw Eng*. 2017;23:1-34.
38. Granger CWJ. Investigating causal relations by econometric models and cross-spectral methods. *Econometr J Econometr Soc*. 1969;11:424-438.
39. Agrawal R, Srikant R. Mining sequential patterns. Paper presented at: Proceeding of the Data Engineering, 1995. Proceedings of the 11th International Conference; 1995:3-14; IEEE.

40. Cai Y, Xiao L, Kazman R. Design rule spaces: a new form of architecture insightPaper presented at: Proceedings of the 38th International Conference on Software Engineering; 2014.
41. Mo R, Cai Y, Kazman R, Lu X. Hotspot patterns: the formal definition and automatic detection of architecture smells. Paper presented at: Proceeding of the 15th International Conference on Software Architecture; May 2015.
42. Xiao L, Cai Y, Kazman R, Mo R, Feng Q. Identifying and quantifying architectural debt. Paper presented at: Proceeding of the 38th International Conference on Software Engineering; 2016.
43. Feng Q, Kazman R, Cai Y, Mo R, Xiao L. An architecture-centric approach to security analysis. Paper presented at: Proceeding of the Proceedings of the 15th International Conference on Software Architecture; May 2016.
44. Lehman MM, Perry DE, Ramil JF. *On Evidence Supporting the Feast Hypothesis and the Laws of Software Evolution*. Washington D.C.: IEEE Press; 1998.
45. Kock N, Avison DE, Baskerville RL, Myers MD, Wood-Harper AT. Is action research: can we serve two masters? (panel session). In: De P, DeGross JI, eds. *ICIS*. Atlanta, Georgia: Association for Information Systems; 1999:582-585.
46. Dutta R, Levine DK, Modica S. Damned if you do and damned if you don't: two masters. *J Econom Theory*. 2018;177:101-125.
47. Cosentino V, Izquierdo JLC, Cabot J. Assessing the bus factor of git repositories. *SANER*. Washington D.C.: IEEE Computer Society; 2015:499-503.
48. Tamburri DA, Lago P, van Vliet H. Organizational social structures for software engineering. *ACM Comput Surv*. 2013;46(1):3:1-3:35.
49. Coelho J, Valente MT. Why modern open source projects fail. Paper presented at: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering; 2017:186-196; ACM.
50. Khondhu J, Capiluppi A, Stol KJ. Is it all lost? a study of inactive open source projects. Paper presented at: Proceeding of the IFIP International Conference on Open Source Systems; 2013:61-79; Springer.
51. Oman P, Hagemeister J. Metrics for assessing a software system's maintainability. Paper presented at: Proceeding of the 1992 Conference on Software Maintenance; 1992:337-344; IEEE.
52. Lee S-YT, Kim H-W, Gupta S. Measuring open source software success. *Omega*. 2009;37(2):426-438.
53. Humphrey WS. Why big software projects fail: the 12 key questions; 2005.
54. Capiluppi A, Lago P, Morisio M. Characteristics of open source projects. Paper presented at: Proceeding of the 7th European Conference on Software Maintenance and Reengineering; vol 2003, 2003:317-327; IEEE.
55. Tourani P, Adams B, Serebrenik A. Code of conduct in open source projectsPaper presented at: Proceeding of the 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER); 2017:24-33; IEEE.
56. Ye Y, Kishida K. Toward an understanding of the motivation open source software developers. Paper presented at: Proceedings of the 25th International Conference on Software Engineering; 2003:419-429; IEEE Computer Society.
57. Avelino G, Passos L, Hora A, Valente MT. A novel approach for estimating truck factors.Paper presented at: Proceeding of the 2016 IEEE 24th International Conference on Program Comprehension (ICPC); 2016:1-10; IEEE.

How to cite this article: Tamburri DA, Blincoe K, Palomba F, Kazman R. *"The Canary in the Coal Mine..."* A cautionary tale from the decline of SourceForge. *Softw Pract Exper*. 2020;50:1930–1951. https://doi.org/10.1002/spe.2874