Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project

Article *in* IEEE Transactions on Software Engineering · September 2015

DOI: 10.1109/TSE.2015.2465387

CITATIONS
19

READS
733

5 authors, including:

Rick Kazman
Carnegie Mellon University
324 PUBLICATIONS **16,207** CITATIONS
SEE PROFILE

Dennis Goldenson
Carnegie Mellon University
57 PUBLICATIONS **2,188** CITATIONS
SEE PROFILE

Ira Monarch
Carnegie Mellon University
76 PUBLICATIONS **1,828** CITATIONS
SEE PROFILE

William Richard Nichols
Carnegie Mellon University
60 PUBLICATIONS **662** CITATIONS
SEE PROFILE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

Evaluating the Effects of Architectural Documentation: A Case Study of a Large Scale Open Source Project

Rick Kazman, Dennis Goldenson, Ira Monarch, William Nichols, and Giuseppe Valetto

Abstract—Sustaining large open source development efforts requires recruiting new participants; however, a lack of architectural documentation might inhibit new participants since large amounts of project knowledge are unavailable to newcomers. We present the results of a multitrait, multimethod analysis of the effects of introducing architectural documentation into a substantial open source project—the Hadoop Distributed File System (HDFS). HDFS had only minimal architectural documentation, and we wanted to discover whether the putative benefits of architectural documentation could be observed over time. To do this, we created and publicized an architecture document and then monitored its usage and effects on the project. The results were somewhat ambiguous: by some measures the architecture documentation appeared to effect the project but not by others. Perhaps of equal importance is our discovery that the project maintained, in its web-accessible JIRA archive of software issues and fixes, enough architectural discussion to support architectural thinking and reasoning. This "emergent" architecture documentation served an important purpose in recording core project members' architectural concerns and resolutions. However, this emergent architecture documentation did not serve all project members equally well; it appears that those on the periphery of the project—newcomers and adopters—still require explicit architecture documentation, as we will show.

Index Terms—software architecture, open source software, documentation

1 Introduction

The benefits of architectural documentation have been long claimed ([14], [39]), but these putative benefits have never been quantified or empirically validated. We have a body of anecdotal evidence for the utility of architectural documentation, but that is all. Since the amount and extent of architectural documentation that should be produced for any given project to reap substantial benefit remains a matter of debate, it is difficult to justify requiring architecture documentation in a project or to make recommendations regarding how much effort a project should expend on it. There are clearly costs associated with the production of architectural documentation. Open Source Software (OSS) communities and agile software development communities tend to emphasize the costs and downplay the benefits, and hence tend to lead developers away from documenting software architecture explicitly. (See the "Manifesto for Agile Software Development" website for more information: www.agilemanifesto.org.) There is no substantive architectural documentation for the vast majority of OSS projects, even the very largest ones. While there is typically code-level documentation in such projects, even this varies greatly in terms of scope and uniformity of coverage [9].

In this paper, we present a hybrid case study in which we studied a widely used OSS project. We pursued the following overarching research goal: to investigate if and how architecture documentation adds value to a large, distributed software project. We did this by observing who used the documentation, for what purposes, and the resulting outcomes. The approach selected for our case study was to establish measurement baselines, introduce the architectural documentation, and then withdraw to collect data. By collecting data from multiple sources we observed different aspects of the project and attempted to triangulate the results to reach conclusions about the value of the introduced architecture documentation. We include a more detailed description of the study design in Section 3.

For our initial step, we created and disseminated a modest set of architecture documentation about and for the Hadoop Distributed File System (HDFS) project—a large, widely adopted, open source project dealing with a complex portion of cloud infrastructure—that had little existing explicit architectural documentation. We captured the following kinds of architectural information in our documentation:

- A module view of the architecture, showing how source files are grouped into packages and the interrelationships of those packages
- a component-and-connector view of the architecture, showing the run-time entities and their relationships

- *Rick Kazman, Dennis Goldenson, and William Nichols are with the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213. Reach them by e-mail at kazman@sei.cmu.edu, dg@sei.cmu.edu, and wrn@sei.cmu.edu.*
- *Ira Monarch can be reached at iramonarch@gmail.com.*
- *Giuseppe Valetto is with the Fondazione Bruno Kessler. Reach him by e-mail at valetto@fbk.eu.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

- an allocation view of the architecture, showing how run-time components are allocated to hardware (This view pre-dated our interaction with the project.)

After we created and disseminated this documentation, we monitored its adoption and analyzed the consequences. In brief, our results are

- The explicit architecture documentation that we created was frequently accessed. Since the website containing the documentation was created, it has been visited more than 25,000 times by more than 17,000 unique visitors, and 33% of the visits were return visits.
- The explicit architecture documentation appeared to have a positive influence on the project, but principally on "outsiders," such as newcomers and less active Contributors and Commenters. These people were not core project members but were likely interested in finding out more about HDFS, to install and use it at their project site, and possibly to tailor it to their own needs. Such users are an important source of innovation that can lead to updates of the HDFS core.
- Members of the project who started out with only marginal involvement—those who had a relationship with the HDFS project in that they had a history of providing comments and bugs—"graduated" to become code Contributors more quickly after the explicit documentation was disseminated.
- After the introduction of the documentation the project's social network became less centralized, in terms of graph-theoretic measures. This hints at a change in knowledge transmission patterns in the HDFS social network. In the absence of (explicitly or implicitly) documented architecture, the only ways of finding key knowledge about the system were reading the source code or asking an expert. The latter technique—asking an expert—tends to increase the centralization of the social network. One possible explanation for the phenomenon we observed is that the existence of explicit architecture documentation alleviated the centralization trend of the social network.

These results can be taken to apply only to large-scale distributed OSS projects since they must, by necessity, manage and coordinate development activities with little or no face-to-face communication. However, this characterization describes a number of important projects, both industrial and OSS. Many large OSS projects—like HDFS—include substantial industrial participation. Smaller projects, projects with fewer team members, and projects where the team members are largely co-located have different communication needs and may not experience the same outcomes from introducing architecture documentation.

While we now believe, based on these results, that architecture documentation does add value and plays an important role in the long-term health of a project, a by-product of our study is that we have broadened our view of the forms that such documentation may take and the processes by which that documentation is created, updated, and used. OSS communities are largely self-organizing. The architectural knowledge for at least this one community—HDFS—appears to have followed this same path. It largely emerged from the communications of key project personnel, rather than following the more traditional model of a centralized, managed artifact. These key project participants, in their day-to-day activities, created a *de facto* body of architectural knowledge that was less formal than, but which served some of the same purposes as, traditional architecture documentation. That is to say, this emergent body of architectural knowledge helped core members with analysis and construction activities. The *de facto* architectural knowledge appears, however, to have been less helpful to outsiders. However, the data we collected about the usage and effect of the *explicit* architectural documentation we created for HDFS are consistent with an educational role, as we will show.

The remainder of this paper is structured as follows: in Section 2 we survey related work on open source communities and architecture documentation and knowledge management. In Section 3 we explain our study design, research questions, case selection, and our data collection and evaluation strategy. Section 4 briefly describes how we documented the HDFS architecture. In Section 5 we describe our empirical results, specifically: i) the data collected on how the architecture documentation was accessed; ii) how the HDFS community changed during the period of study, including promotion data and communication patterns; iii) a textual analysis of the project's technical communications before and after the introduction of the architecture documentation; iv) an analysis of project performance indicators; and v) a survey of HDFS Contributors and Committers. Finally, in Section 6 we summarize our findings, discuss threats to validity, make recommendations based on our findings, and sketch our future work.

2 Related Work

The benefits of having a well-defined software (or system) architecture have been claimed for decades. For example, Bass et al. describe 13 arguments in favor of the importance of architecture [8]:

1. An architecture will inhibit or enable a system's driving quality attributes.
2. Architecture decisions allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.
8. An architecture can provide the basis for evolu-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

tionary prototyping.

9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that forms the heart of a product line.
11. Architecture-based development focuses attention on the component assembly.
12. By restricting design alternatives, architecture reduces design and system complexity.
13. An architecture can be the foundation for training a new team member.

Other authors (e.g., Taylor et al. [67]) have made similar claims. These include claims of, and in a few cases evidence for, organizational benefits, technical benefits, and social benefits. As we will show, the HDFS project's communications (principally its JIRA archive) manifest many of these uses of architecture. That is we see, in the recorded discussions among the project Contributors, most of these concerns being aired and addressed.

There has been considerable research investigating the uses of architectural information: how it is created and managed, how it is disseminated and shared, and its effects on the project (e.g., [5]). However, the bulk of this work has focused on a "knowledge management" perspective, where knowledge is centrally created and disseminated in a traditional (typically closed-source) project context.

More generally, there has also been considerable research in studying communication in OSS projects based on a large literature of socio-cognitive analyses of language use and participant interaction in scientific and engineering projects (e.g., [10], [45], [31], [41], [11], [30], and [58]). Strands of these analyses are brought together in a 2006 paper, "A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software" [56],[1] where it is observed that *"OSS projects … rarely rely on explicit system-level design, or on project plans or schedules. Rather, OSS developers work in arbitrary locations and collaborate almost exclusively over the Internet, using simple tools such as email and software code tracking databases (e.g., CVS – Concurrent Versioning System)."* Obtaining architecture information in ways other than through an explicit and authoritative architecture document is corroborated by other studies where it is shown that it can be obtained via OSS project blogs [52], mailing lists [29], social media [70], forums, code comments, and commit data [69]. Our study found similar results using text analytic methods on thousands of records in the HDFS issue database that also enabled us reach beyond our survey results to help determine whether the architecture documentation we introduced to HDFS had an effect on discussions of its participants who did not respond to the survey. These text analytic methods were based on the work of Callon et al. [10], Cambrosio et al. [11], and Coulter et al. [15]. However, instead of applying the methods to detect potential conceptual, structural and practical changes at the level of a technical domain like polymer chemistry, software engineering or biological safety research, the methods were applied to a single software project,[2] in this case an open source one, to detect possible conceptual, structural and practical changes based on the new architectural documentation.

Another paper by Mockus et al. [47] that also considers OSS communication and interaction, even discussing architecture and documentation somewhat, though not as the main focus, has some methodological similarities to ours. It reports their investigation of Apache and Mozilla, two of the most studied OSS projects [64]. The study used email archives of source code change history and problem reports to quantify aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution intervals for the OSS projects. A hypothesis of theirs that they found some confirmation of and that is relevant to our study is that *"If a project is so large that more than 10-15 people are required to complete 80% of the code in the desired time frame* [which seems to be true of HDFS], *then other mechanisms, rather than just informal, ad hoc arrangements, will be required in order to coordinate the work."* Some of our findings seem to indicate that one of these mechanisms may be architectural documentation.

There have been few studies that have assessed the uses of architecture in open source software (although Babar offers some hints in this direction [5]), and none, to our knowledge, that have systematically examined the *influence* of architectural documentation. There are some studies that while not focusing solely on OSS projects, do include them [68], and their findings are similar to studies of architectural documentation in commercial or closed source projects [54] as well as findings of the one recent study that does focus solely on OSS software architecture documentation [22]. All these studies agree that architecture documents are often out-of-date, inconsistent or of low quality, provide inadequately for specific task and context, and do not support useful ways of interacting with the documentation in terms of search and navigation. However, there seems to be some disagreement between two of the studies concerning the use of architecture views, viewpoints, and mappings among views. While Rost et al. [54] argue that most industrial developers are dissatisfied with architecture views and viewpoints, Ding et al. [22] cite surveys like the one by Malavolta et al. [46] that find about 85% of architects from industry use multiple views for architectural description, and 59% of them

[1] The methodological framework described in an article by Sack et al. [56] for studying the heterogeneous components of a project's hybrid network is similar to the one used in our study in that it combines ethnography, text mining and socio-technical network analysis to understand OSS development. Moreover, the documents they analyze are enhancement proposals similar to the issues records accessible in the HDFS Web database that we use. However, they do not consider the potential impact that explicit architecture documentation might have on OSS development communication and interaction.

[2] As had been done in other text analysis work in software engineering (e.g., [48]).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

use mechanisms to ensure cross-view consistency. In any case, for OSS projects, Ding et al. [22] find that architecture documents in OSS development pay little attention to employing views (7.4%) and viewpoints (0.0%). Moreover and perhaps most importantly they also report that just 5.4% of open source projects have *any* software architecture documentation at all. Scacchi and his co-authors have examined scores of open source projects, and have found that not only do these projects lack documented architectures, but they even lack documented requirements [1], [59]. They do note, however, that requirements and architecture *emerge* from an ongoing discussion among key project participants. While we do not dispute these findings, our objective is to determine whether when architecture documentation that contains views and viewpoints is introduced into an OSS project, it will *influence* participant discussion and work and whether it will thereby make a valuable contribution.

Osterlund and Crowston [51] have explored the use of documents in open source and open content communities, specifically focusing on how documents serve collaborators with symmetric and asymmetric access to knowledge versus documents. Their primary finding is that documents supporting asymmetric groups (such as Contributors and Committers) are likely to be more prescriptive and explicate their own use, when compared with documents supporting symmetric groups. Although Osterlund and Crowston looked at bug reports, commit messages, and patches, their results shed some light on the utility of architectural documentation, as we will show.

Finally, the empirical analysis we have carried out can be construed as an attempt to assess how well HDFS has done as a project, before and after the introduction of explicit architectural documentation. There has been substantial research about how to define and measure OSS project success. Crowston, Howison, and Annabi [18], among others, have observed how that is a complex and multi-faceted matter. For instance, some literature in the early and mid-2000s focused on the *maturity* of OSS projects and the software product they develop, with the main intent of providing guidance on OSS adoption to potential industrial and commercial customers. Several OSS maturity frameworks have been proposed to that end, including OMM [53], OSMM [28], QSOS [4], OpenBRR [50], and OpenBQR [66].

Other works have tried to assess OSS projects' success in terms of the *validity* of its output (i.e., the software produced). Validity can be expressed through quality measures, which can be applied to the software directly (*intrinsic validity* criteria), or by measures of the value and satisfaction with the product, as perceived by its Contributors or users (*extrinsic validity* criteria). A recent review of OSS empirical research [19] observes that "*the majority of the empirical work uses code quality measures"* to assess the intrinsic validity of a project's output. It lists about 20 works that use such code-level indicators to evaluate software qualities such as maintainability, defect density, usability, and reliability. Design-level methods for assessing OSS validity, which are relevant to the theme of our own research, are scarcer. For example, in 2006, MacCormack et al. describe a technique to quantify system modularity and assess its variations and improvements over time, and apply it to very prominent OSS products like Linux and Mozilla [44]. In a 2011 paper, Wong et al. propose an automated approach that considers together design dependencies and version change histories to enumerate possible modularity violations that can be detrimental to the software quality, and use it to evaluate some OSS products [73]. To our knowledge, the literature lack works describing architecture-level measures of the intrinsic validity of OSS products. As per extrinsic validity, it is common to cast the value of an OSS product with *popularity* metrics, such as satisfaction rating, and usage or download counts, which focus on value as perceived by adopters and users of the software, like in a 2009 paper by Lee et al. [42]. Other studies, like the one conducted by Subramanian et al. in 2009 [65], consider also the satisfaction of the developers involved in the project.

A different view of OSS success tries to assess the *sustainability* of a project as a continuing and productive initiative. In 2007, English & Schweik took a collective action stance and looked at OSS software as a "public good"; therefore, an OSS project cannot be successful, unless it is able to attract—and then retain—a vibrant group of collaborators, who remain committed to making regular progress and sustain software maintenance [24]. The paper operationalized this sustainability idea with four metrics: project age, number of releases, regularity of releases, and number of downloads. Based on those metrics, an OSS project—either at a young, initial stage, or in a later, more mature phase—can be classified as a success, failure (i.e., subject to abandonment or likely to be abandoned), or as having a yet indeterminate fate. Later, Wiggins and Crowston revisited that classification, proposed some adaptations to those sustainability metrics, and explored their validity as reliable predictors of success [71]. Whereas studies like the one conducted by English and Schweik in 2007 [24] judge an OSS project only based on its sustained outcomes, other works also consider the sustained activity of the project community, measuring the amount and distribution of work ([47], [26], [57]), or volume and characteristics of communication ([72], [76]), since they vary from time to time. Many of these metrics are easily observable proxies of the dynamics and vitality of the community, since they can be derived directly from the open project repositories that characterize OSS. A special case of these kind of studies consider the *health of the community* undertaking an OSS project as a self-evident measure of its success, and propose ways to measure such health. A significant number of works propose methods from the discipline of Software Network Analysis as a means to assess health, and predicate on the structure of the project social networks, and its evolution patterns. For instance, a 2007 paper by Long and Siau [43] considers the transition from a hub network structure to the well-known core-periphery onion structure ([47], [17], [37]) as a sign of a maturing project; in 2010, He et al. analyzed how centrality influences patterns of collaborations of new OSS members [32]; whereas in

2013, Zanetti et al. discuss the effect of network centralization on quality of bug reports and their resolution likelihood [77].

In our own work, we have used several of the approaches above, with the intent of triangulating among different definitions and ways to measure OSS success. Some of the measures that we have used and that we will discuss in the remainder of this paper speak to the validity of the software being produced, others speak to the sustainability of the project over time, and others still to the health of the community undertaking it.

3 Study Design

The overarching goal of the research was to investigate if and how architecture documentation adds value to a software project. Since we could not measure value directly, to investigate this question we investigated the observable changes in participant behaviors, the concepts that participants use in their discussions, and explicit project outcomes. To accomplish this we had to address a number of constraints. First, we recognized that we would have limited access to the project's participants: Contributors and Committers. Second, we could not exercise control over the project or project environment, with the one exception of introducing the documentation. Third, because of limits on our funding, we not only had to complete data collection within approximately one calendar year, but we would also be unable to follow up on emerging opportunities. We expected to have limited access to project participants for direct questions, but we expected to have full access to the project archives, including source code, records of change requests, discussions, and patch history.

Given these limitations, we decided to use an embedded mixed method [16] case study [75] approach. This approach includes collection of data from multiple sources, at multiples times while retaining the features of a single case with embedded units of analysis [75]. For Creswell, mixed methods collect and combine qualitative and quantitative data, where the embedding design has to be either qualitative or quantitative but not both, and the embedded design has to be primarily what the embedding design is not. For this research, the embedding design is a case study in the sense of a descriptive study of a process where the relevant behaviors cannot be manipulated, so no causal claims can be made. We adopted this approach because it can provide deeper understanding of the phenomena under study, even if it does not provide statistical evidence for causal inference [55]. All the methods involving quantitative analysis are embedded in our case study. An overview of our approach follows.

After producing and introducing the architectural description document, we studied the amount that the document was used and later surveyed project participants for their views on, and understanding of, architecture. We later determined whether the architecture document could have had an impact on software development practices by analyzing whether there was a change in the kinds of terminology project participants used to discuss software issues and how to resolve them. We determined whether the architecture documentation could have had an impact on the aggregate behaviors of participants by including outcome measures of patches, communication patterns, and participation roles before and after the introduction of the documentation. To compensate for the inherent limitations of case studies from simple pre/post designs, we determined to collect the project's social network structure, as well as patch and role activity as time series data [13]. These threads of data were largely independent and concurrent throughout the study. The exceptions to concurrency were that the survey was administered partway through our study, and the text analysis of participant's discussions was added later in the research. These multiple threads of data were then combined to synthesize a deeper understanding of how the project outcomes connected to documentation use and the concepts used by participants.

3.1 Research Questions

We formulated eight research questions to address the observable uses of the documentation, the effects on project performance, and the social network dynamics of the participants. We also developed the questions and protocol to examine the text from project communications, as well as a survey of project members to determine their knowledge of and attitudes toward architecture documentation. The research questions we formulated were

RQ 1.1 Was the architecture document read and if so, how much and how did it change?

RQ 1.2 Was the architecture document referred to by the project Contributors and Committers?

RQ 2.1 Was the introduction of the architecture document associated with a change in submission activity?

RQ 2.2 Was the introduction of the architecture document associated with a change in the quality of submissions?

RQ 3.1 Was the introduction of the architecture document associated with faster promotion from Commenter to Contributor to Committer?

RQ 3.2 Was the introduction of the architecture document associated with changes in project communication patterns?

RQ 4.1 Were there any measurable differences in the use of architectural concepts in discussion of issues before and after the architecture document was introduced?

RQ 5.1 How did the Contributors and Committers use the key concepts outlined in the architecture document?

3.2 Case Selection

To address the research questions, we required a project that could meet several important criteria:

- The project should address architecturally important quality attributes, specifically performance, security, availability, and modifiability but

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

should not have an explicit architectural documentation.
- To provide sufficient participants for analysis requires a moderately large project, with more than 10 or so core members and more Commenters, as well as a large community of users.
- To generalize beyond open source, the project should be large enough and important enough to have significant industry support and collaboration.
- The project's communication must be observable and preferably recorded in an accessible collaboration tool.
- Data on project evolution must be available.
- A project leader must be motivated to participate by collaborating with the documentation and providing support reaching the community.

For these reasons we selected the HDFS project, which met all our criteria. The Apache Hadoop project is widely used by large companies such as Yahoo!, eBay, Facebook, and others.[3] The lowest level of the Hadoop stack is the Hadoop Distributed File System [60]. This is a file system modeled on the Google File System and is designed for high volume and highly reliable storage [27]. Clusters of 5,000 servers and over 5 petabytes of storage are not uncommon within the HDFS user community.

3.3 Data Collection and Evaluation Strategy

We just stated that we were interested in determining if architecture documentation adds value to a project. Quantitative and observable measures of value include improved *project outcomes* such as (1) increased participation, (2) increased productivity and (3) improved quality. We agreed that these could be measured using the counts of Committers, submitters, and dates of submittals and commitments before and after the introduction of the architecture document. These changes should also be associated with changes in the *behavior* of the participants. To plausibly associate project outcomes with the architecture document, we merged changes in concept usage of Contributors and Committers with both changes in document usage and observable changes in the behavior of project participants. For document usage we analyze the records of access to the architecture documentation website that is provided by Google Analytics. To measure changes in concept use we analyzed textual records of project communications as well as survey findings. To measure the centrality and centralization of the communications among participants, we analyzed project communication patterns. We made these measures by analyzing time series pre- and post-intervention.

Thus we evaluated our research questions by

- Tracking how often the architecture documentation is downloaded and how often it is mentioned in discussion groups (addressing RQ 1.1 and reported in Section 5.1).
- Tracking whether any changes have occurred in the interactions and activities of the HDFS developers—Contributors and Committers (addressing RQ 2.1 and RQ 2.2). We did this by examining changes to their social network, as represented by their communications on mailing lists and on posted JIRA issues (reported in Section 5.2).
- Tracking project community health measures, such as the growth of the committer group, and the time lag between someone's appearance as a Contributor and their acceptance as a Committer and other measures (addressing RQ 3.1 and RQ 3.2 as reported in Section 5.2).
- Tracking whether the introduction of the architecture documentation changed how the project community discussed the system (addressing RQ 4.1). That is, whether architecture concepts as represented in the introduced architecture documentation were used more (and by more people) in their discussions after the introduction to the documentation (reported in Section 5.3).
- Tracking product performance indicators, such as project capacity—how often Contributors made submissions to the system—and submission quality—how often submissions were rejected by the Committers (addressing RQ 2.1 and RQ 2.2, reported in Section 5.4).
- Surveying the HDFS Contributor and Committer community on their opinions of the value of the architecture documentation that we created (addressing RQ 1.2 and RQ 5.1 and reported in Section 5.5).

4 Documenting the HDFS Architecture

When writing architectural documentation it is necessary to have an overview of the major system components and abstractions, and how they interact. When there is a single architect for the system, the easiest route is to simply talk to the person. Most open source projects, however, do not have a single identifiable architect since the architecture is typically the shared responsibility of the group of Committers [47].

Architectural documentation is believed to serve three major purposes: (1) providing a means of introducing new project members to the system, (2) serving as a vehicle for communication among stakeholders, and (3) being the basis for system analysis and construction [14]. But architectural documentation, to be truly useful, must also connect design concepts to their realization in the code. This connection was missing in the existing, minimal architectural documentation on the Hadoop website. But someone who wants to become a project Contributor or Committer needs to know which modules to modify and which are affected by a modification. Communication among stakeholders regarding proposed changes is also typically couched in terms of the effects on code units.

Thus we set as our task to create architecture documentation that not only captured the main abstractions employed in HDFS, but also to connect those abstrac-

[3] For a list of Hadoop users, see the Apache wiki at http://wiki.apache.org/hadoop/PoweredBy

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

tions to the code (files) that developers work on every day.

4.1 Gaining an Overview

The first step of our documentation process was to gain an overview of the system. Subsequent steps included elaborating, validating, and refining the documentation. To do this we turned first to published sources and then we interviewed one of the key project Committers.

Prior to our intervention, the only explicit architectural documentation on the Apache Hadoop website [3] briefly described some of the *run-time* concepts used in HDFS. This document provides a single architectural diagram, as shown in Figure 1. The main run-time components in HDFS are the ***NameNode*** that manages the HDFS namespace and a collection of ***DataNodes*** that store the actual data in files. High availability is addressed by maintaining multiple copies of each data block in an HDFS file; by detecting corrupted data blocks or failed DataNodes; and by mechanisms that can automatically replace a failed DataNode or a bad data block. But for system construction, maintenance, and evolution to proceed, the code units and their responsibilities must be unambiguously identified. The lack of such architecture documentation may be an impediment to the efficient evolution of a non-trivial system, and to the efficient education of newcomers. Furthermore, having such documentation might assist Contributors become Committers faster. We explore these research questions—RQ 3.1 and RQ 3.2—in Section 5.2.

Architectural documentation occupies a middle ground between (relatively abstract) architectural concepts and code and it connects the two. Creating this explicit connection is what we saw as our most important task in producing the architectural documentation for HDFS. Furthermore, as we will discuss in Section 5.4, there was shared architectural knowledge, but the mechanism for sharing was email and JIRA postings, which were unstructured and informal.

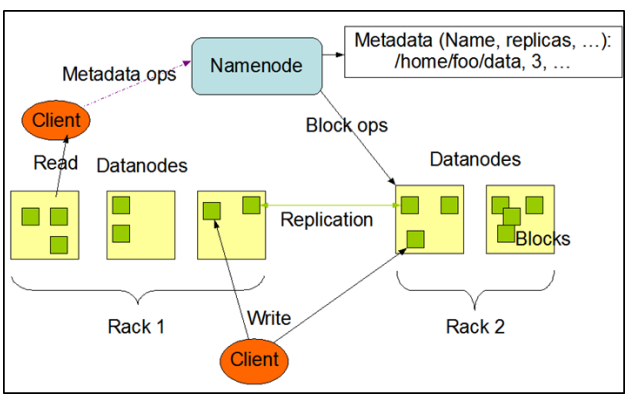

Fig. 1. HDFS Run-Time Concepts

4.1.1 Expert Interview

To elicit the architecture we first conducted a face-to-face interview with one of the key Committers of HDFS (as described in a 2011 paper by Bass et al. [7]). In this interview, we elicited a number of architectural views, including a description of the major HDFS modules and their interactions. This allowed us to correlate the existing architectural concepts—as shown in Figure 1—with the code modules that a developer would see.

4.1.2 Tool Support

An interview was a good starting point to identify the major architectural concepts, but it did not result in authoritative and unambiguous information. The technique we next used to create an authoritative set of architectural documentation was reverse engineering. We used two commercial tools—SonarJ and Lattix—which aid the analyst in creating architectural representations, starting from a static analysis of code dependencies [62], [40]. Neither tool provided satisfactory representations, however. For example, Figure 2 shows a view of the most important code modules of HDFS, along with their dependency relationships, as generated by SonarJ.

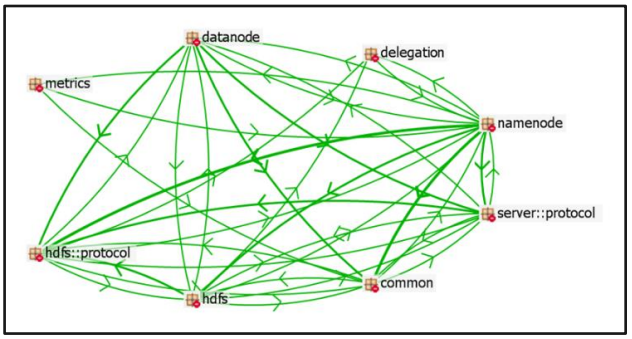

Fig. 2. Reverse-Engineered Module Relationships in HDFS

It appears, at least on first analysis, that the HDFS modules form a highly connected graph. We postulate that this reflects a major concern for the project. Modularity affects the understandability and modifiability of a system. Performance and availability were, however, the most important quality attributes of the HDFS community, and these concerns—not modifiability, portability, or evolvability—shaped the architectural structure.

But SonarJ and Lattix aid an analyst in determining the modular and layered structures in the architecture by supporting the definition of design rules and visualizing where the project's modular structure violates the rules. Modifiability, while important, has thus far been addressed in the HDFS project by keeping the code base at a relatively modest size and by having a substantial number of Committers devote a non-trivial amount of time learning the code base. Most of the Committers are commercial developers paid by their employers to work full-time on HDFS.

This lack of (architectural) attention to modifiability is a potential risk for the project as it grows: it makes it difficult to become a new Contributor since the learning curve is quite steep. Hence a goal of our research was to determine whether the addition of architectural documentation could make a noticeable different in the ease with which a newcomer became a useful and productive Contributor to the project.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

The documentation was published at a publicly accessible location[4] and we published this location on the Hadoop wiki.[5] We also advertised the publication of the architecture documentation via Hadoop's JIRA. We focused on documenting two views of the architecture: a run-time (component-and-connector) view, similar to Figure 1, and a module view, as shown in Figure 3. A third (allocation) view already existed in the project's documentation. Finally, we tracked and evaluated the usage of the published architecture documentation, as we describe in Section 5.1.

5 Empirical Results

Following the study design and methods described in Section 3, we investigated the effects of the introduction of architecture documentation to the HDFS project by means of a multi-pronged empirical study. In this section, we describe the details of the techniques that we used to analyze the project and report on the gathered empirical data and the corresponding analytical results, which are categorized as follows:

- Data about the usage of the HDFS architectural documentation that we produced:
    - Data about the consultation of the HDFS documentation (made available through Google Analytics reports on the corresponding Web pages)
- Data about the effects of the architectural documentation on the HDFS project:
    - Effects of the documentation on the discourse of the HDFS community and individuals in the community (determined via surveys and a textual analysis of project technical communications)
    - Effects of the documentation on variations in the structure and dynamics of the HDFS community (determined via a social network analysis of the project technical communications)
    - Effects of the documentation on variations in performance factors of the HDFS project (extracted from the project's JIRA issue tracking database)

A significant amount of the above data collection and analysis originates from the project repositories used in daily HDFS development work; in particular, the HDFS JIRA issue tracking database, the developer mailing list archives, and the project's SVN code repository.

From the JIRA archive, we are able to gather information about HDFS releases and details about individual JIRA "issues" (which amount to development tasks) within those releases. This data includes the properties related to an issue (e.g., the reporter, assignee, date opened, description, severity), the issue's entire change history, comments within it, and contributed patches attempting to close the issue.

The developer mailing list archive gives us access to details about communication trends within a time period. We are able to store and analyze the complete text of individual messages and message threads, including message subject, contents, sender address, recipients, date, and reply targets.

The final data source that we use is the SVN code repository for HDFS. Here we are able to see which files are changed within a commit, the details of how those files have been changed, who does the committing, and when the commits happen.

To automate both data collection and data analysis from those project repositories, we have developed a series of reusable tools built upon the Taverna Workflow Management System (www.taverna.org.uk). These tools, known as *workflows*, target a specific data repository for the HDFS project and collect all data from that repository, usually within a specified date range.

Data collection workflows harvest data from the repositories and store them in a relational database of our design, where relationships between different data types from disparate sources can be established. For instance, we resolved aliases, names, and email addresses across all three data sources to individual actors that have a role in the project. We also used clues within the data that we collect to bridge the different repositories and provide traceability among them. For example, the JIRA issues were tied to SVN commits by examining the revision number supplied when an issue was resolved. Additionally, mailing list messages were tied to JIRA issues by examining the contents of the subject and message body. They were also linked to releases by examining the dates that communication threads span. By linking our data, we enhanced the accuracy of our views of the project, its actors, and their interactions across releases of the HDFS software.

[4] http://kazman.shidler.hawaii.edu/ArchDoc.html

[5] http://wiki.apache.org/hadoop/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

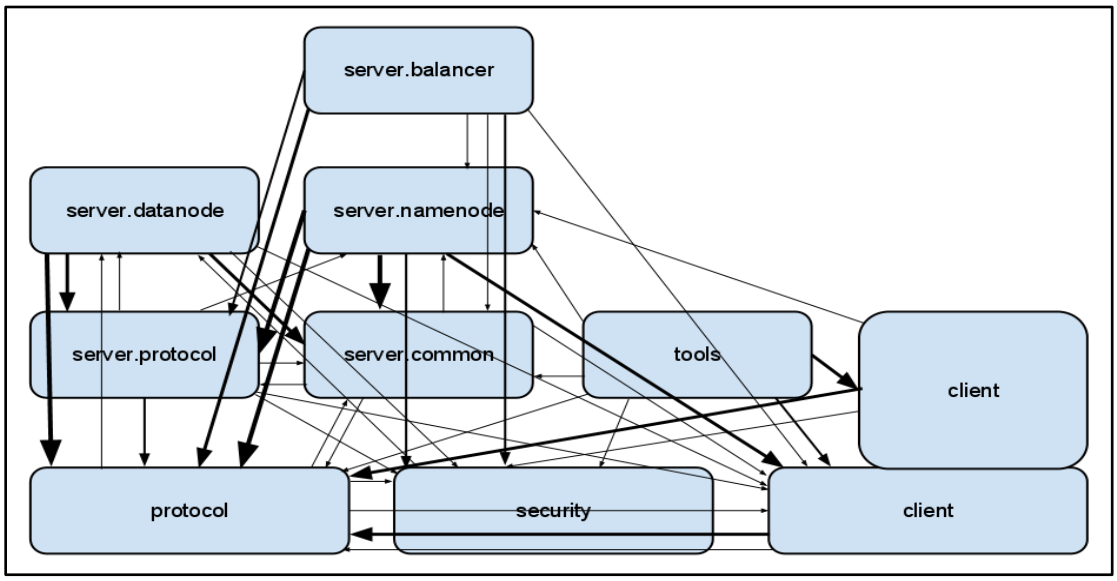

Fig. 3. Documented Module Relationships in HDFS

5.1 HDFS Architecture Documentation Raw Access Data

As mentioned above, the HDFS architecture documentation was made publically available via a website and a link to this site was placed on the Hadoop wiki. In creating this documentation and making it publicly accessible, we wanted to know if the community—users, Contributors, and Committers—was actually accessing, reading, and returning to the documentation over time. To help evaluate RQ1.1 the usage of the architecture document was tracked using the Google Analytics suite of tools.[6]

5.1.1 Data Analysis Procedure

Google Analytics helps track a wide variety of measures of a website's usage. The measures that we primarily cared about monitoring were: total page visits, returning visits, bounce rate, and time spent on each page. Figure 4 shows a screen capture from Google Analytics between August 1, 2013 – March 4, 2015.

5.1.2 Discussion

In the 19 months portrayed in Figure 4, the architecture documentation website was visited over 14,000 times by more than 10,000 individuals, and almost 30% of these visits were from return visitors. Furthermore, the usage has steadily increased over time, from just over 300 visits per month in June of 2011 to almost 1,000 visits per month currently, which suggests that the greater HDFS community found the architecture documentation and found it to be sufficiently useful to prompt substantial numbers of return visits. As of the writing this paper, the architecture documentation website still exists and has had more than 25,000 visitors, over 8,000 of whom are *return* visitors.

Clearly, someone is using this documentation and a sizeable minority of all visits is finding the architecture documentation useful enough to return to the site. Furthermore, the site is being accessed world-wide. Figure 5 shows the geographic distribution of site accesses. Therefore, the answer to RQ 1.1 is yes the architecture document was accessed, a sizable number of the accesses were return visits, and the access rate increased steadily during the study period.

5.2 HDFS Community Analysis

To evaluate RQ 3.1 and RQ 3.2 we examined the dynamics of roles and participation among contributors to the HDFS project. The objective of this analysis was to consider whether such dynamics may have been influenced by the introduction of the architectural documentation. People operating within the HDFS community (as with most open source communities) participate in a variety of roles. We focused on the following roles:

- **Commenter**: The individual posts in the project repositories, namely JIRA and mailing lists, one or more comments about change requests, work items, bug and issue reports, or other tasks.
- **Contributor**: The individual submits one or more actual software contributions in the form of proposed patches, in the context of an open JIRA issue.

  Committer: The individual has obtained commit rights and submits one or more approved changes to the code base of the project, in the context of an open JIRA issue. Only a limited subset of developers have such commit rights.

Any HDFS participant may have multiple roles. The Contributor role subsumes the Commenter role; that is, it is typical (although not necessary) that a Contributor has acted as a Commenter at least at some point during the project timeline. Analogously, a Committer role subsumes the Contributor role. Consistently with prevalent "onion-like" models of open source organizations, such as those

[6] http://www.google.com/analytics/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

described by Mockus et al. [47], Crowston and Howison [17], and Kazman and Chen [37], these three roles represent increasing levels of participation and higher status in the project. We therefore postulate a "promotion chain" between the roles:

Commenter → Contributor → Committer

We also observe "promotion events," which occur when a Commenter submits his or her first patch, or when a Contributor commits code in the HDFS code base for the first time. Note that not all the HDFS participants enter the community in the Commenter role and work their way up the promotion chain; several individuals show up for the first time as Contributor or even Committer. Also notice that it is possible (although rare) for a Commenter to be promoted directly to Committer.

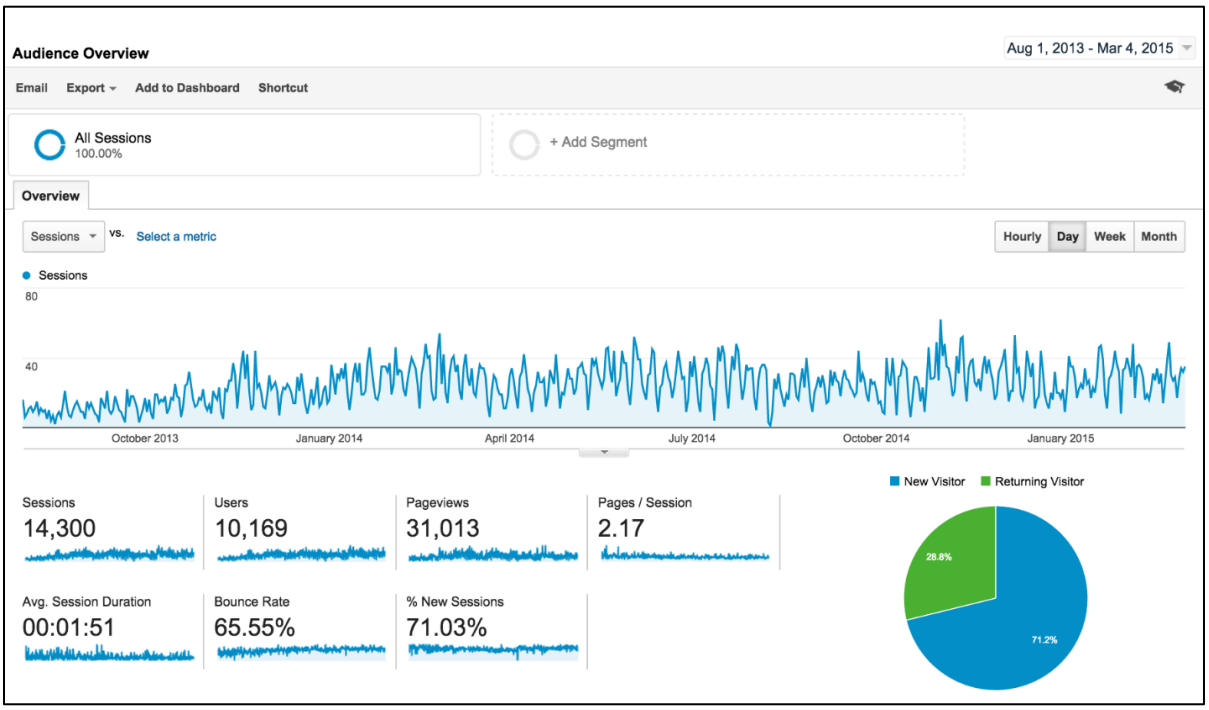

Fig. 4. Visitors to the Architecture Documentation Website

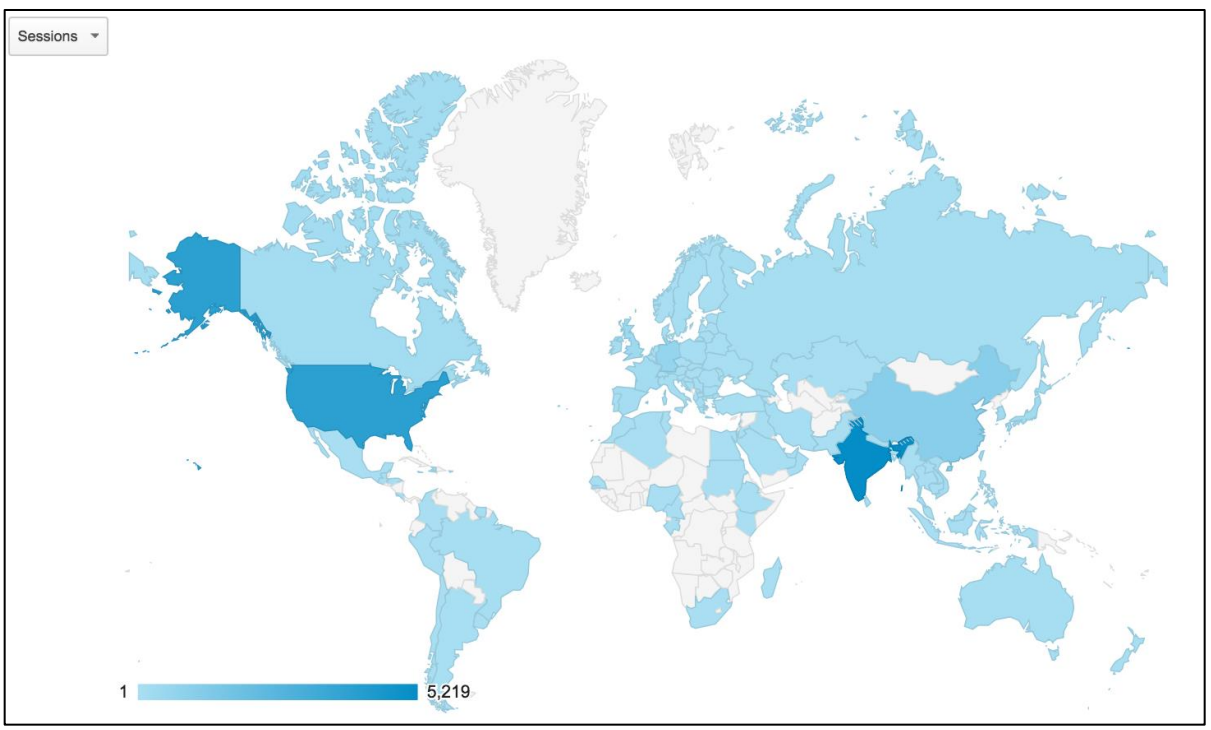

Fig. 5. Geographic Distribution of Architecture Documentation Website Visitors

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

For the HDFS community analysis, we have collected for each person the following record from the project repositories:

> [Unique ID, ROLE, First Comment Date, First Contribution Date, First Commit Date, Length of Comment to Contribution Period, Length of Contribution to Commit Period, Length of Comment to Commit Period, Count of Comments before First Contribution, Count of Commented Issues before First Contribution, Count of Contributions before First Commit, Count of Issues Contributed to before First Commit , Count of Comments before First Commit, Count of Commented Issues before First Commit]

Note that the **ROLE** field contains one of the three classifiers {Commenter, Contributor, Committer} based on the evidence collected in the record. We classify each participant with the classifier for the highest role in the promotion chain for which we have evidence. For example, if the participant with ID 1366 has a First Comment Date field of July 4, 2011, empty First Contribution Date field and an empty First Commit Date fields, she is classified as **Commenter**; if participant 531 has a First Comment Date of December 17, 2009, and a First Contribution Date of August 30, 2011, but an empty First Commit Date, she is classified as **Contributor**; if participant 1925 has an empty First Comment Date field, as well as an empty First Contribution Date field, but a First Commit Date of November 13, 2010, she is classified as **Committer**.

We have collected 444 such records, which we call the "promotion data set," covering traces of project participation from May 7, 2006 (early HDFS development within the umbrella of the Hadoop project), to February 29, 2012. This period includes the date at which HDFS became a standalone Apache project, after its spinoff from the Hadoop project on June 19, 2009, as well as the moment in May 2011, when we introduced our HDFS architectural documentation.

From the HDFS repositories we have also collected a "communication data set," that is, communication data attributed to all participants, from their messages posted on JIRA or the project mailing lists in the same time span. The population of participants covered by this communications data set is the same as the population from the promotion data set, and covers several thousand messages. We have used the communications data set to construct a social network of the community, in which the nodes are individuals, and the arcs are directed and weighted, representing person-to-person acts of communication. The method used to construct this social network is the same one adopted in a paper by Ehrlich and Cataldo in 2012 [23].

We have carried out two separate analyses on the data described above, one on the promotion data set and another on the communications data set.

5.2.1 Data Analysis Procedure

The first analysis of the promotion data set investigates the hypothesis, connected to our RQ 3.1, that the architectural documentation may have facilitated *newcomers* to the HDFS community in contributing more effectively to the development effort. This hypothesis postulates that if knowledge is concisely summarized in an architecture document, then it becomes easier for a newcomer to learn the architecture, in contrast with a situation in which knowledge is casually transmitted, and must be gleaned by consulting via email lists, JIRA archives, and so forth. We define newcomers as people entering the community as Commenters and who spend one or more days as Commenter before moving up the promotion chain.

To explore this hypothesis, we looked at two groups: newcomers who entered the HDFS community after the introduction of the architectural documentation, vs. newcomers who entered the HDFS community at any point in the project history earlier than June 1, 2011. We carried out several statistical tests to see whether the progression in status is different for those two groups of newcomers.

The second analysis focuses on the social network derived from the communications data set, and how structural metrics of that social network change over the course of the project. The hypothesis we have investigated, which is connected to our RQ 3.2, is that the introduction of the documentation may have changed the *pattern* of communication and information sharing in the HDFS community, and, because of that, impacted the structure of the social network and/or the position of individuals in the network.

To investigate that hypothesis, we looked in particular at **centrality** and **centralization** metrics. Centrality is a positional metric that applies to individual nodes; a more central position in the network indicates the importance or influence of the person in the HDFS community. Centralization is a structural metric of the entire network: the centralization score of a given network expresses how similar that network is to a theoretical maximally centralized network, such as, a *"hub-and-spoke"* network.

There are several different definitions of node centrality, each with its own interpretation and semantics. Each centrality metric yields a corresponding centralization metric for the entire network. Among the various flavors of centrality, we have selected *information centrality* [63]. In contrast with other well-known definitions of centrality, such as *betweenness centrality* [2], [25], which only consider *geodesics* (that is, the shortest paths between pairs of nodes information centrality), information centrality considers *all* paths, weighed inversely to their length, to identify brokers and compute an index of their importance between 0 and 1. Because of its definition, information centrality is considered a good metric to assess the centrality of an individual in self-regulating organizations that are flexible and informal—like an OSS community—where paths of communication are not engineered or consistently enforced, but rather emergent and self-organized. It is also a good choice in situations where information is not necessarily propagated by one-to-one direct communication but rather via fora where it remains persistent and available to anyone for consultation. This is definitely true for discussions of JIRA items and mailing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

lists in the HDFS project. Similarly, the global measure of *information centralization* over the whole social network [63] is suitable to capture how much information and know-how is held centrally, as opposed to diffused throughout the same kinds of organizational settings.

For all the participants in HDFS we have computed information centrality every 14 days, obtaining a time series with 153 data points, 20 of which cover the period after the publication of the architectural documentation. From those values, we calculated a similar time series for the information centrality of the HDFS community.

5.2.2 Results of the Analysis of the Promotion Data Set

First, we tested the difference in proportion for the "Length of Comment to Contribution Period" datum. This datum measures how *quickly* a newcomer, who starts her experience in the HDFS community as a Commenter, finds herself in the position of being able to submit some code to the community, and therefore graduates to the Contributor role in the promotion chain. For our test, we considered the following groups:

- Group1: newcomers who entered the project in the role of Commenter at any time earlier than June 1, 2011 and have become Contributors within a 274-days time span (n=50)
- Group2: newcomers who entered the project in the role of Commenter on or after June 1, 2011 and have become Contributors in the 274 days-period after that date and up to February 29, 2012 (n=11)

Group1 is more than four times larger than Group2, because for Group1, we consider a much longer segment of the project history, which yields a much larger population of newcomer Commenters. Among them, there is a significant number of newcomers who were promoted to Contributor, but only after more than 274 days. However, since we did not have visibility into how long it could take to achieve promotion for newcomers who entered as Commenters *after* June 1, 2011, and had not yet been promoted by February 29, 2012, we have decided—for the sake of fairness—to include in Group1 only those who achieved their promotion within the same 274-day time span we could observe for Group2.

The basic statistical characteristics of the two groups are reported in Table 1. The table shows that Group1 has higher median and mean values for the "Length of Comment to Contribution Period" datum, and that its values are more widely dispersed than those of Group2. To evaluate the significance of this observation, we carried out a Mann-Whitney test between Group1 and Group2, as defined above, with the hypothesis that "Length of Comment to Contribution Period" would be significantly less for Group2. The meaning of this hypothesis is that Group2 is facilitated to move up the promotion chain more quickly, and in particular can become able earlier than Group1 to produce code that is submitted for evaluation and quality control to the project. The results of the test **(W = 161.5, p-value = 0.017)** support our hypothesis with a good degree of statistical significance.

Graphically, the difference between the two groups is visible from the boxplot in Figure 6, which is drawn on a logarithmic scale. Each boxplot shows the data range between the 1st and the 3rd quartile of the two data sets as a box, with the median values marked within the boxes as thick black lines. The "whiskers" that extend outside of the boxes extend by convention 1.5 times the inter-quartile range above and below the box; it is common to consider values beyond the whiskers as outliers, and is often customary to exclude such values from statistical tests of significance; however, in this case, all the values for both Group1 and Group2 remain within the whiskers range; all have hence been included in the Mann-Whitney test.

The results of this test suggest that new participants entering the HDFS community after the introduction of the architectural documentation may have an easier path in the project, as they produce their first software patch and graduate to the role of Contributor significantly quicker. To further investigate this hypothesis, we tried to see whether other facets of the newcomers' participation could be attributed to this quicker promotion rate. One factor influencing promotion could be the intensity of participation in the community by newcomers in the Commenter role. To consider that, we carried out a Mann-Whitney test of difference in proportion between the same groups, for the "Count of Comments before First Contribution" datum, which can be seen as a proxy for the intensity of the participation in the community. The result **(W = 221.5, p-value = 0.3125)** shows no statistically significant difference between Group1 and Group2 in this respect. The interpretation of this result is that the quicker promotion rate for people in Group2 cannot be attributed to their being more active or enthusiastic HDFS Commenters than newcomers who entered the same community in the past.

TABLE 1

Statistical Comparisons of Group1 and Group2 – Promotion Time in Calendar Days

	N	Min.	1st Quartile	Median	Mean	3rd Quartile	MAX	Std. Dev.
Group1	50	2	13.75	56	82.12	132	269	76.08
Group2	11	2	7.5	27	28.36	37.5	80	26.59

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

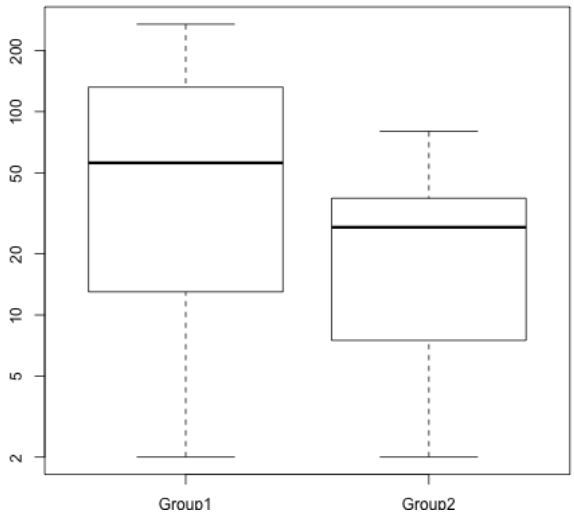

Fig. 6. Difference in Proportion for Datum Length of Comment to Contribution Period

5.2.3 Results of the Analysis of the Communication Data Set

We examined the measure of information centralization for the social network representing the cumulative interpersonal technical communication within the HDFS community. We sampled information centralization every 14 days starting on May 21, 2006, thus obtaining a *time series* with 153 data points; the last 20 data points are measures of information centralization taken *after* the introduction of the architectural documentation until February 29, 2012.

The hypothesis we explored using this data is whether the architectural documentation has contributed to make the social network of HDFS less centralized. A less centralized network indicates that information and know-how (and their flow) are less concentrated within a relatively small set of "power players." This would be consistent with a beneficial impact of the architectural documentation in promoting a more even access to, and diffusion of, knowledge within the HDFS community, and would thus be consistent with—and reinforce—the statistical findings from the analysis of the promotion data set presented in the previous subsection.

To explore this hypothesis, we analyzed the time series of information centralization data, which is shown in Figures 7 and 8. This time series has an increasing trend, and the random fluctuations seem to stabilize to a close-to-constant amplitude since the second half of 2008; it also has a small seasonal component. To evaluate the effect of the architectural documentation on the trend of information centralization over time, we constructed two Holt's exponential smoothing models from the time series, after removing the seasonal component [33]. The first model, which we call the "early model," is constructed on the basis of only the first 133 data points, that is, the information centralization measures observed before the introduction of the architectural documentation; the second model, which we call the "complete model" uses all the 153 data points.

A useful property of Holt's models is that they can be used to forecast values ahead in the future, based on the trend of the time series. For that purpose, we used the forecasting algorithms offered by the R package *forecast* v.3.0.3 [34]. First, we carried out a forecast using the early model, for the 20 14-day periods elapsed from the introduction of the architectural documentation on June 1, 2011 all the way to February 29, 2012; the result of this forecast is displayed in Figure 8, which shows a forecasted trend (red line) that is about constant. We then carried out another forecast for the same 20 14-day periods, on the basis of the *complete* model. The difference is that the complete model had been constructed incorporating the knowledge of the actual values of information centralization for those periods. Thus its forecasts take into account that additional information. The forecast obtained this way is displayed in Figure 8, which shows a clear downward slope for the forecasted trend for the information centralization of the HDFS social network.

To better appreciate the difference between the two forecasts, we plotted them in scale in Figure 9: the red line represent the information centralization values forecast on the basis of the early model, while the blue line represents those forecast on the basis of the complete model.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

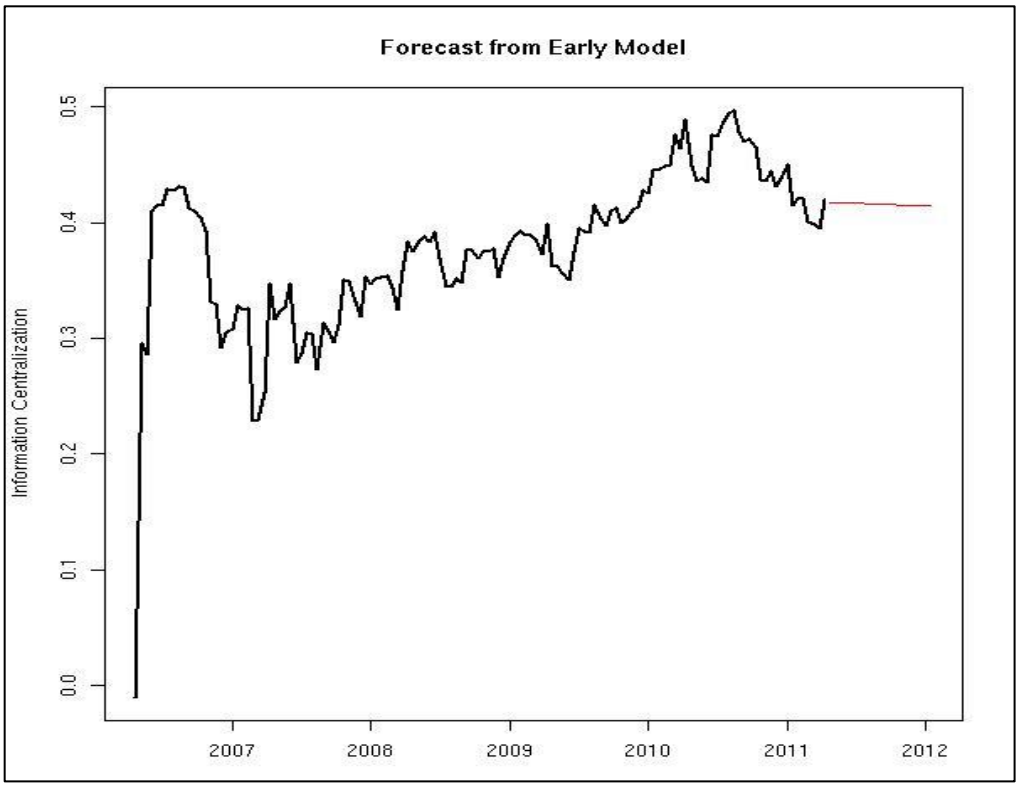

Fig. 7. Information Centralization Forecast from the Early Holtz Model

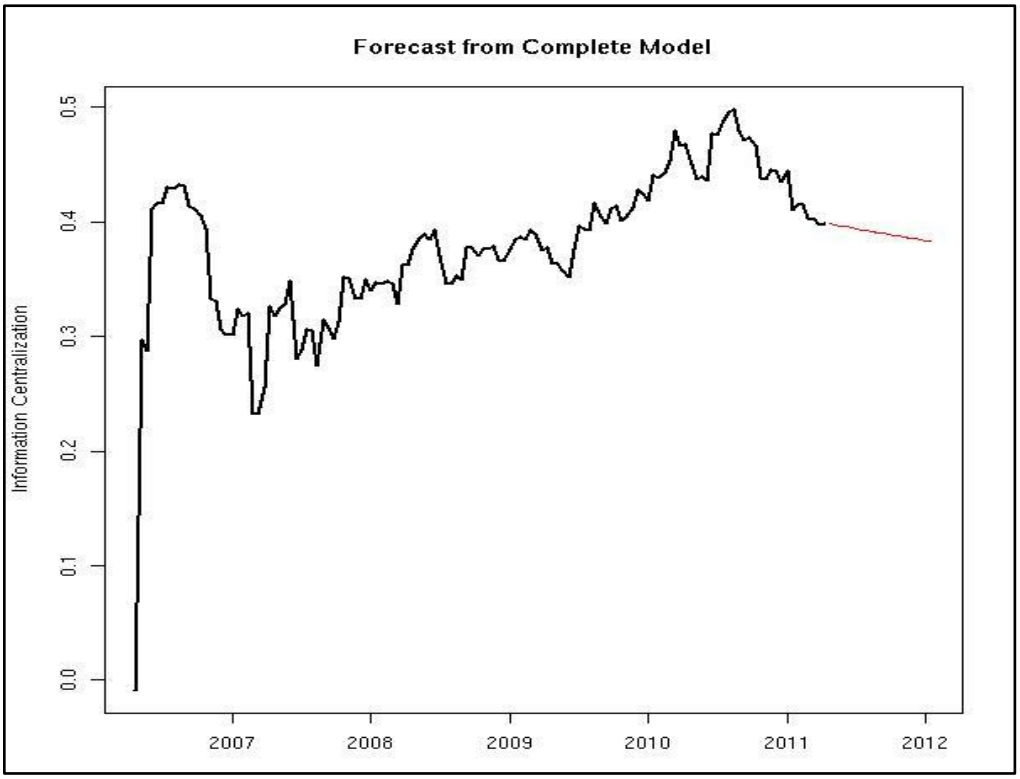

Fig. 8. Information Centralization Forecast from the Complete Holtz Model (Full-Time Series Data Set)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

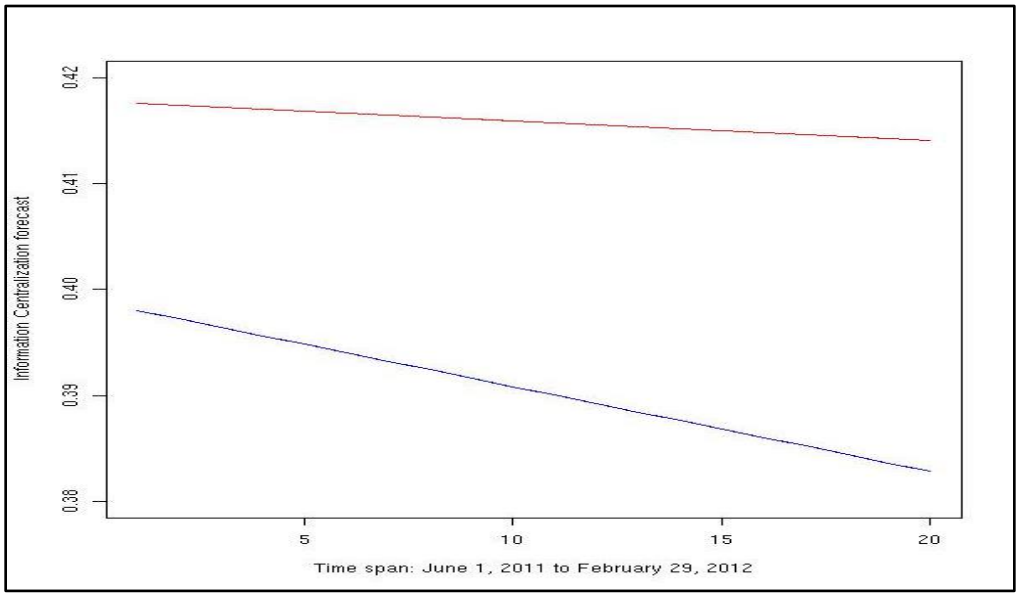

Fig. 9. Forecast Comparison: Early Model (Red) vs. Complete Model (Blue)

An analysis of the accuracy of the two forecasts, via their Root Mean Square Error (RMSE), shows that the complete model is slightly better, as should be expected. Also other measures of forecasting error are congruent with RMSE, providing us with confidence that the complete model is indeed more accurate.

$RMSE_{early\ model} = 0.024$

$RMSE_{complete\ model} = 0.022$

The interpretation of these results is that the information centralization of the social network of the HDFS community trended towards becoming less centralized after the introduction of the architectural documentation, which is a result consistent with the hypothesis that the architectural documentation may have contributed to sharing technical know-how in a more diffused way. It is worth noticing that a lower level of information centralization is a trait that tends to favor the knowledge acquisition process of *newcomers*, by making them less dependent on connections with "power players" in the network, and providing them with more nodes and paths they can leverage in their knowledge quests.

5.2.4 Discussion

We have presented a multifold statistical analysis of community-related data mined from the HDFS repositories. The analysis of the promotion data set offers evidence supporting a positive answer to RQ 3.1 on the role of architectural documentation in facilitating promotion within the OSS community of HDFS. Similarly, the analysis of the communication data set offers evidence supporting a positive answer to RQ 3.2 on the more widespread diffusion of architectural know-how.

These results, which originate from two disjoint data sets, reinforce each other; they are both consistent with the overarching hypothesis that the introduction of the architectural documentation has had significant effects on the organizational dynamics, albeit modest in magnitude. Moreover, those measured effects both point in the direction of a facilitating role of architecture documentation especially for newcomers in the community. The statistical tests we have carried out on the promotion data set suggest a benefit for individual participants, who may become empowered to provide constructive contributions to the HDFS project more quickly. The statistical tests on the communication data set suggest also a benefit to the HDFS community as a whole, in that the know-how necessary to participants to be effective Contributors may have become more diffused.

5.2 Textual Analysis of Technical Communications

To answer RQ 4.1 we analyzed the textual descriptions of several thousand HDFS issues and their resolutions. The software issues were taken from the publically accessible JIRA database with descriptions, summaries of issues and comments on the issues over their lifetime. We also looked at records derived from the JIRA database describing commits to resolution of issues and another database of HDFS email threads. The information in the records describing commits did not differ significantly from the description and summary information in the publically accessible database so their analysis is not reviewed here. The email messages focused on programming details and were not informative of the higher level issues involved, so their analysis is not reviewed here either.

The data from the four databases is divided into two time periods:

- March 2010 through May 2011
- June 2011 to February 28, 2012

There are 950-1000 issues for each time period.

5.3.1 Data Analysis Procedure

The purpose of the textual analysis was to determine whether

1. Concepts used in the discussion of HDFS issues are architectural and, if so
    a. How frequently and in what proportions

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

are they used?

b. Who authorizes their introduction and usage?

2. The HDFS architecture documentation produced by one of the authors[7] is relevant to, or can be elaborated to become relevant to, actual past and ongoing issue discussions.
3. The architecture documentation has impacted these discussions and concepts by changing
   a. the frequency or portions of architectural concepts used
   b. the extent to which articulation work is centralized or decentralized.

As part of our analysis of HDFS issues we produced a *concept map* that shows relationships among the most significant concepts used in issue discussions. The concepts and relationships mapped are *prima facie* deemed architectural to the extent that they correspond to the dependencies and other relationships described in the HDFS architecture documentation. To the extent that they do correspond, the architecture documentation will also be deemed relevant to the discussion of HDFS issues. If the correspondence increases after the architecture documentation's introduction in June 2011, this would suggest that the documentation has impacted the discussions.

The analysis of the issue descriptions was done with the help of a text analysis tool called Leximancer [61]. The tool enables rapid analysis of thousands of issues but also allows modulating the results through researcher intervention and interpretation. Words and phrases are clustered automatically into affinity groups, each represented by single terms called *concepts*, and concepts in turn are also clustered automatically into higher level abstractions called *themes*. An affinity group for a concept includes the terms whose usage in the text corpus is more similar to one another than to other terms not in the group. Many of the terms in the affinity group are not themselves concepts identified by the analysis.

Concepts are named by the term in the affinity group that has the highest total similarity score with the other terms in the group. Also, concepts belonging to the same theme are more similar in usage than concepts not belonging to the theme. A completed automated analysis produces a concept map, as shown in Figure 10.

The map displays concepts (represented by labeled dots) laid out in a two-dimensional space where proximity between dots represents similarity of usage and size of dots represents frequency of concept occurrence. Lines link concepts that co-occur in the same descriptions. By default, a map only shows links between a given concept and others that are in its proximity up to a given threshold and that have a higher probability of co-occurrence than others in this vicinity. (See the concept **DataNode** [in black] at the middle top of Figure 10 and its links to the concepts **client, GS**, **rack** and **block**.) Links between more distant concepts (based on frequency of co-occurrence alone) can be shown at a researcher's discretion. Clusters of concepts that are similar in meaning are enclosed in circles. These clusters represent themes and are labeled by the most representative concept in the cluster just below the concept **DataNode**. (See the ***DataNode***[8] theme, colored red in Figure 10.)

For the automated analysis to produce results that include concepts of special interest to the research, terms standing for these concepts must occur in the texts being analyzed and have relationships to other terms expressing other concepts also found in these texts. For example in Figure 11, quality attribute concepts like **availability** near **DataNode** and **scalability** and **performance** near **NameNode** would not have appeared unless designated as terms of interest, but the fact that they showed up as concepts near **DataNode** and **NameNode** respectively means they are more relevant to these concepts than others in the map.

Concepts can also be grouped within or across themes after the automated analysis is completed by encircling them within themes or across themes without rearranging the concepts or modifying their links. (See Figure 11, where the grouping includes the themes ***failover***, ***recovery***, ***NameNode***, ***performance***, ***fails*** and ***scalability***.)

5.3.2 Results of the Analysis

Our analysis revealed concepts and relationships corresponding to those described in our architectural documentation. The documentation describes both a run-time (component-and-connector) view of the HDFS architecture in the Overview, Communication, and Use Cases sections and a module view of the architecture in the Module View section. The Decomposition section of the Documentation is relevant to both these views. Component-and-connector architectural concepts are among the highest ranking concepts in terms of frequency and linkage. However a module view is also evidenced in the concept maps,[9] though less prevalently.

Concepts relevant to component-and-connector or module views are architectural not only because they correspond to the key concepts described in the architectural documentation but also because they are associated with run-time quality attributes in the case of the component-and-connector view (e.g., **performance**, **availability**, and **scalability**)[10] or concepts important to modularization in the case of the module view (e.g., ***refactoring***, **classes**,

[7] This HDFS architecture documentation captured the major architectural decisions of HDFS 0.21 and provided a guide to the overall structure of the HDFS code [7].

[8] Names of concepts are in bold. Names of themes are in bold italic.

[9] In Figure 10, concepts in the themes ***refactor***, ***user*** and ***method*** are indicative of a module view, and in Figure 11, concepts in the themes ***methods***, ***dependency*** and ***build*** are also indicative of this view. These themes and concepts are, however, much less prevalent than those indicating a component-and-connector view.

[10] In Figure 10, **scalability** and **performance** are linked to **NameNode** and **availability** is in the theme ***DataNode***. In Figure 11, **availability** is contained in the overlap of themes ***failover*** and ***NameNode*** and the theme ***performance*** overlaps with the themes ***NameNode*** and ***fails***. **Scalability** also links with **performance**.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

and **packages**).[11]

Both component-and-connector and module view concepts and relationships can be seen in the analyses both before and after the introduction of the architecture documentation. The frequency and centrality of these concepts were also similar in the two time periods. Moreover when terms and affinity groupings corresponding to quality attributes were added to the analysis, they appeared in concept clusters and linkages associating the respective views with the quality attributes they are expected to address.

Similar patterns occurred in both time periods. For example, although there were cases in which refactoring is explicitly mentioned in both time periods, they were almost always concerned with Java classes rather than modules. The kind of module refactoring suggested in the architectural documentation was not found in issue records either before or after the introduction of the documentation.

Given the above findings, there is much to suggest in answer to RQ 4.1 that there were not any measurable differences in the use of architectural concepts before and after the architecture document was introduced. There is therefore little to suggest that our architecture documentation was associated with any *changes* in the concepts used in HDFS participants' discussion of issues and their resolutions at least as recorded in their issue database. Very telling is that, as far as the discussion of issues is concerned, there is no reference to the architecture document by project Contributors and Committers (re RQ 1.2). Moreover, although there is evidence for component-and-connector and module architectural views being used to reason about issues, these views are quite similar before and after the introduction of the architecture documentation.

Nevertheless, regarding the concept **package,** there is some evidence that provides some measurable indication of a change in how architectural concepts were used in issue discussions. As described in our documentation, the use of packages embodies the module structure of HDFS. In the time period March 2010 –to May 2011, before the introduction of our architecture documentation, there are just 7 references to **package** and only one case that might be construed as relevant to modularization. Much of the content concerns what is private or public in packages. After introduction of the architecture documentation, there are 32 references to **package** and several cases where relevance to modularization is more explicit, though there is still much discussion of private and public packages. The difference in frequency with which "package' is used in the later time period and the more explicit concern with modularity in some cases could suggest changes in concepts after introduction of the architecture documentation in the later time period.

In addition to analysis of concept maps before and after the introduction of architectural documentation, questions of the architecture documentation being associated with changes in project communication patterns (RQ 3.2) can be addressed from the point of view of the frequency and proportionality of architectural concepts among different HDFS groups. Two groups are of particular interest: (1) those who are referenced by name in JIRA discussions at least 20 times in the description of issues, typically Committers and Contributors, and (2) those who were not named in JIRA discussions, but only appear in the Assigned-To or Submitted-By fields and who are not Committers. We refer to the former as Heavy Hitters (HHs) and the latter as non-Heavy Hitters (non-HHs).[12] The number of HHs varied but was always under 20. Non-HHs numbers also varied but were usually around 70.

Measureable indicators to determine increased participation and change in project communication patterns are based on group differences focusing on the *percentage* of a person's architectural concept frequency against their total concept frequency. An average was calculated for each group. The averages remained similar for HHs before and after introduction of the architectural documentation even while the amount of commenting increased. However, the architectural percentage for the non-HHs in the latter time period increased more and was even a little higher than the architectural percentage for the HHs in this later time period.

Another point to consider concerns the findings for the amount of discussion being done by the HHs. The number of issues went up from 989 issues in the initial time period to 1056 issues in the latter time period, even though the latter time period was considerably shorter. Total (including architectural) concept frequency was higher for HHs before introduction of architecture documentation, but more so after. This is why the HHs are holding much more architecture discussion than the non-HHs, especially in the later time period. This increase in sheer amount of HH discussion and decrease in non-HH discussion, suggests perhaps a change in communication pattern in the direction of greater centralization of HHs rather than less. However, since non-HHs were saying less, but saying it more architecturally, perhaps they were working more economically.

[11] In Figure 10, ***refactor*** is a theme and the concept **refactor** is linked to **class. Package** is a concept in the theme ***refactor***.

In Figure 11, **package**, **refactor**, **class**, **public**, and **private** are clustered together in the ***methods*** theme.

[12] HHs may be one way of operationalizing core participants, and non-HHs may be a way of operationalizing peripheral participants.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

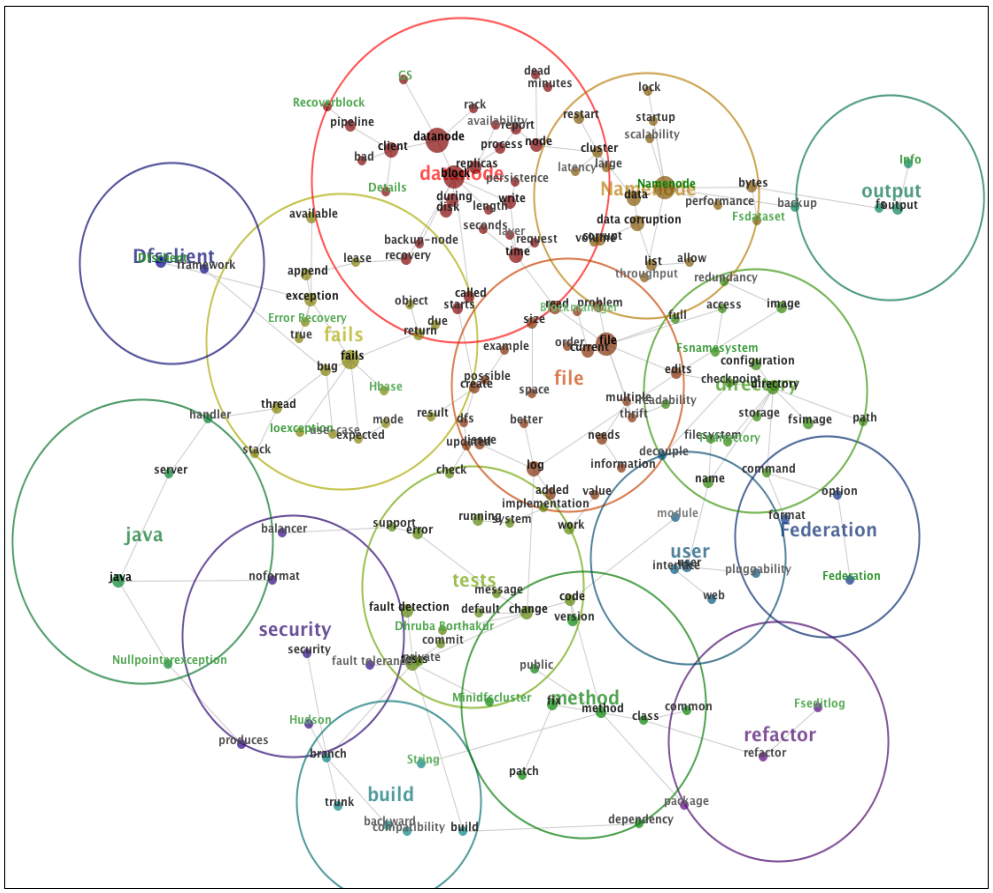

Fig. 10. JIRA Concepts March 2010 - May 2011

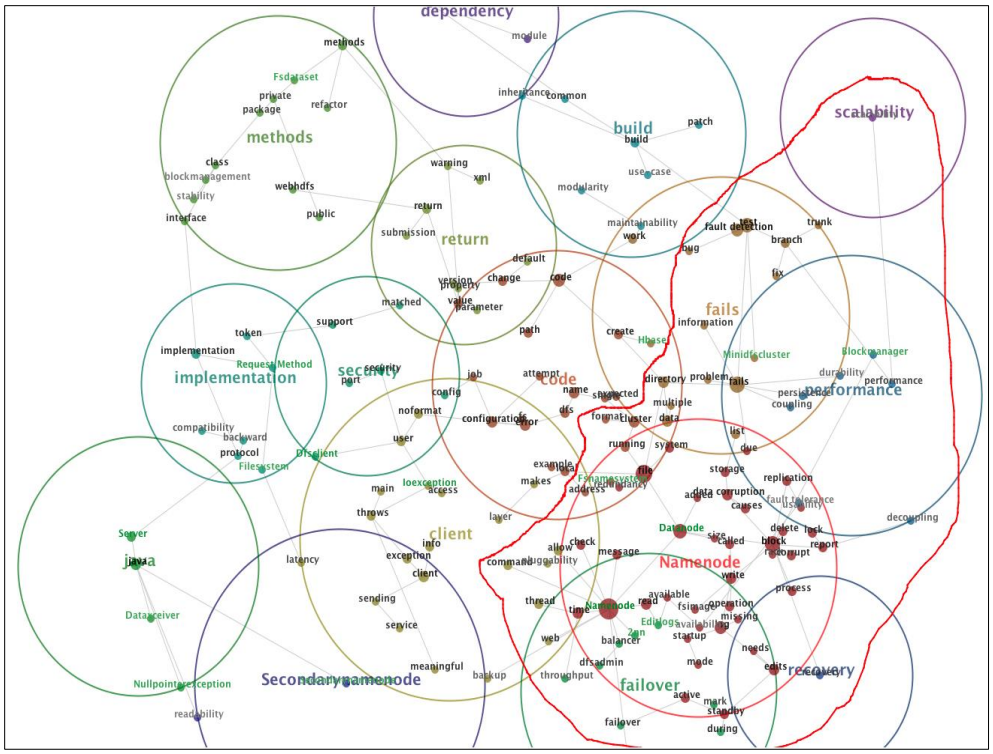

Fig. 11. JIRA Concepts June 2011 – February 2012

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

5.4 Discussion

Based on the architectural documentation produced for HDFS, the conceptual map findings and the findings for HHs/non-HHs, it is clear that architectural concepts and reasoning were used in JIRA issue discussions.[13] Moreover, while most of these discussions were geared to fixing bugs or making low-level software improvements, 14% to 20% of the discussions *involved* architectural concerns. Even though text analysis did not suggest that introduction of the explicit architectural documentation was associated with a change in HDFS discussions (since architectural discussion stayed largely the same before and after its introduction), it did show that architectural concepts and reasoning *did* occur and were influential in both time periods via the JIRA discussions as well as increasing percentage-wise for non-HHs.

Moreover, the reasoning appeared to be divided into two architectural views exhibited by two thematic clusters and their relationships. According to the analysis, the much more prevalent component-and-connector view was used in describing issues and understanding how they came about so they could be fixed. The much less prevalent module view was exhibited in discussions of refactoring and the use of packages. However, no specific modules were identifiable in the concept maps. Interestingly, one person's name was overwhelmingly associated with packages in June 2011 – March 2012 and one person was more associated with refactoring than anyone else in both March 2010 – May 2011 and June 2011 – March 2012. The module view indicated by the analysis was limited to only a few HDFS developers.

In addition, because some concepts and relationships in thematic clusters could be mapped to concepts and relationships in the HDFS architectural documentation, the documentation was obviously relevant to issue discussions. In fact, the mapping of architectural concepts and relationships from one to the other can be taken a step further. Text analysis of issue discussions shows that the discussions themselves can be a source for sharing architectural concepts and using them to reason architecturally about issues without having to rely on explicit architectural documentation.

Text analysis, especially with respect to differences between HHs and non-HHs, showed somewhat more centralization of articulation work rather than less after the introduction of the architecture documentation. However, there was some indication that non-HHs devoted a little more discussion to architecture in the later time period than they had in the earlier time period and this *could* be due to their use of the architecture documentation. After all, having a single concise document to peruse is easier than searching numerous JIRA discussions when it is not known what search terms to use.

[13] In a series of papers, the latest of which was written by Dalle and David in 2007 [20], a stochastic simulation modeling tool is used to study the implications of the mechanisms by which individual software developers' efforts are allocated within large and complex open source software projects. To produce these models, Dale and David had to *suppose* that when developers consider contributing to a given project at a given moment in time, they are aware of an emerging software architecture "whose respective technical properties and state of development (at that moment) are fully known." We have shown here not only that such an architecture exists for HDFS, but also what its actual properties and states are and how the emerging architecture is captured in issue discussions.

5.4 HDFS Project Performance Indicators

The objective of introducing the architecture document into the project was to increase the number of participants and increase their capacity to make changes. The proposed improvement mechanism was that the explicit documentation reduces the cost of entry and improves understanding, thereby increasing submission capacity, improving the quality of submissions, and reducing the rate of rejections. This portion of the research focused on determining if the expected outcomes were realized.

We structured the analysis as a pre-post intervention design without controls. The research questions addressed in this analysis were as follows:

RQ 2.1 Was the introduction of the architecture document associated with a change in submission activity?

RQ 2.2 Was the introduction of the architecture document associated with a change in the quality of submissions?

The following sections describe how these questions were investigated.

5.4.1 Data and Variables

The events we examined were user-contributed patches to open issues. We chose data derived from patches as proxies for participation, activity, and submission quality. The patch events include submitted and accepted (that is committed by the Committers) or rejected. We extracted project patch activity data from the SVN source code repository. For each event, we noted the date, issue addressed, and the submitter.

We examined the number of issues addressed with at least one patch; there may be multiple patches associated with an issue. The issues were categorized by project personnel into critical, blocking, major, and trivial. To separate overall counts from those deemed most urgent, we chose to examine all issues, as well as looking at just the critical and blocking issues. For each of these issues, we counted the submitted patches, those accepted, and those rejected. We used the ratio of commits and rejections per patch as a measure for submission quality, with the assumption that higher quality submissions are more likely to be committed than rejected.

To measure throughput, we counted patches submitted and accepted patches per period. To measure quality of patches, we used the ratio of rejected patches to total patches submitted. Finally, we analyzed the number of issues resolved and the number of critical and blocking issues resolved.

5.4.2 Data Analysis Procedure

We segmented the data into two distinct time periods, before and after introduction of the architecture documentation as a pre/post design. We aggregated data over two

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

weeks each for each period. To control for calendar effects on effort applied, the periods after the introduction were matched to comparable dates from the previous year. This gave us 21 periods each for two data sets.

First, we applied a paired-t test to matched calendar periods before and after the introduction and computed the descriptive statistics and correlation coefficients. Second, we applied the equivalent non-parametric Wilcoxon test to confirm that the assumptions of the parametric test were not terribly violated. Finally, we plotted the data as time series in two week intervals and computed the pre and post trends and correlation coefficients using regression.

5.4.3 Results of the Analysis

The run charts for issues and patches before and after the introduction of the document are shown in Figure 12. The total issues trend-line shows little slope and little change before and after the introduction. The patches, however, trend negatively in both time frames, but increase substantially at the time of introduction.

The results of the parametric fits are summarized in Table 2. The 21 time periods post documentation were more active than the time periods prior. The numbers of issues addressed with at least one patch, patches submitted, committed, and rejected all increased by statistically and practically significant amounts. The mean values of issues patched, patches submitted, commits, rejects, and critical and blocking issues resolved per period had statistical significance of $p < 0.05$ and totals increased by about 80% after introduction of the architecture document. The ratios of rejections or commits per issue were only modestly changed with a p value of 0.58. The ratio of rejection/commits increased by about a third with a p value slightly greater than 0.05. The correlation between pairs at comparable times of the year was weak. The largest correlation between paired time periods was R=0.38, many were very small or negative.

5.4.4 Discussion

Although the observed pre/post differences in patches submitted, committed, and rejected was significant, we doubt a causal relationship. We observe that the correlation between paired time periods was very weak for patches and even weaker for issues, suggesting that the increase was not uniform, but highly variable. Nor did contribution activity depend strongly upon the time of year. We expected that if the document was effective at increasing contribution activity, the increases would be more consistent across comparable calendar periods. Although the aggregate production was higher after the introduction of the document, this factor was unable to predict an increase in comparable time periods before and after.

The timing of the increase in activity also raises doubts. The activity begins to peak immediately prior to the document introduction and reaches a new peak in the initial few weeks. Overall activity then trends downward, though with high variation. We expected, rather, that the document effect would be lagged and increasing. A direct effect should lag as developers discovered the document, read the document, and started to use the document. Instead, patch activity peaked early and decreased over time. We infer that there other substantial uncontrolled factors affecting the developer activities more strongly than vacation seasons, holidays and other calendar effects, and the document.

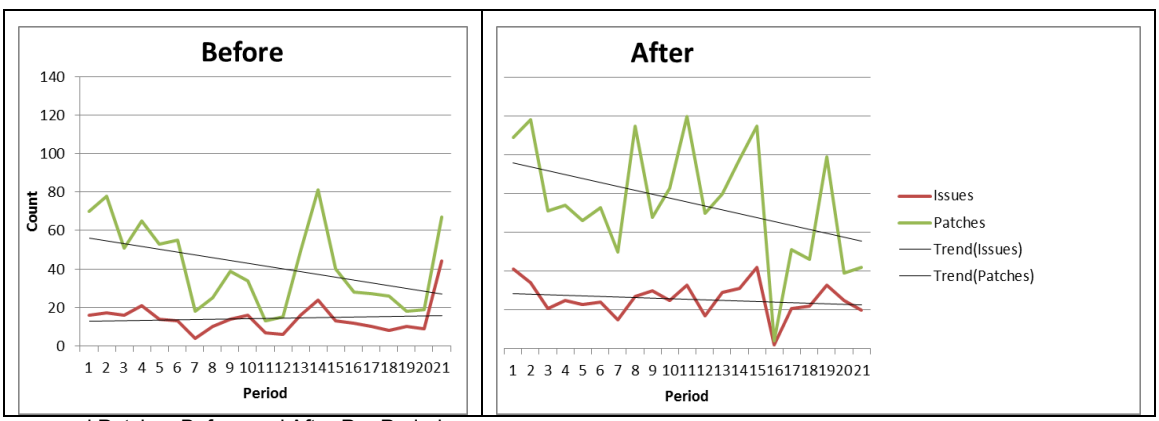

Fig. 12. Issues and Patches Before and After Per Period

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

TABLE 2. PARAMETRIC FITS

Per period	Mean		Standard Error		p	r
	Before	After	Before	After		
Issues	14.3	25.7	1.83	1.94	0.0002	0.06
Patches Submitted	40.1	74.1	4.76	6.86	0.0002	0.17
Commits	28.3	49.1	3.15	4.87	0.006	-0.18
Rejects	17.2	35.4	2.63	3.85	0.0002	0.38
Commits per issue	2.04	1.91	0.13	0.14	0.58	-0.51
Rejects per issue	1.25	1.34	0.15	0.12	0.58	0.32
Rejects/Commits	0.757	0.997	0.088	0.105	0.051	-0.05
Issues Resolved	19.4	29.9	3.98	2.42	0.067	-.039
Critical and Blocking Issues Resolved	1.71	3.76	0.737	0.697	0.022	-0.12

The two week periods were chosen to reduce seasonal effects, but do not account for other special causes such as the release schedules, patches driven by business needs, and new product users. Sample size and intrinsic variation can be managed with statistical techniques; the special causes, however, are more problematic. Any project changes during the period prior to or after the introduction confound the analysis. Unfortunately for the research, the special causes cannot be easily controlled and it is undesirable for the project to avoid them. Although we considered, after-the-fact, searching for ways to statistically control for these inappropriate effects, we preferred to use our case study methodology and user surveys to provide insights.

Another reason to reject the conclusion that the document had a direct effect was that we found no measured effect size in the *quality* of the submitted patches as measured by the rejection ratio. We expected developer access to the architecture documentation to improve the quality of submissions or increase the completeness of a Committer's review. No statistically significant effect could mean either that the effects cancelled (i.e., no net effect) or that there was no effect at all. The ratio trends negative, but variation remained very high. Confounding factors could include new and less experienced developers or more difficult problems being addressed. This analysis, therefore, does not find evidence for improvement in quality.

In summary, there appear to be underlying trends and a great deal of variability in the results that indicate a number of other factors to be considered. With respect to RQ 2.1, although patch activity increased, we doubt the association with the introduction of the documentation without other more compelling evidence. With respect to RQ2.2, the quality of submissions appears to have decreased post introduction, but again we doubt the association with the document; rather this seems to be dominated by confounding factors and high variation.

5.5 Survey of HDFS Contributors and Committers

To address RQ 1.2 and RQ 5.1, we decided to collect survey data when it became apparent that additional insights about the HDFS architecture could be gleaned by asking for direct feedback from those who worked most closely with it. We also decided to field the survey to capture information about the extent to which the HDFS Contributors and Committers found value in the architectural guidance document that we prepared. We conducted two parallel surveys, one for HDFS Contributors and another for Committers.[14] The questions focused on architectural and implementation areas where the survey respondents had historically experienced difficulty in implementing their patches. Both questionnaires also queried the respondents about the sources of guidance they consult in preparing their patches and completing their tasks, including a short series of questions about our architectural guidance document. We also asked the Committers about the kinds of errors made by the Contributors they reviewed and the criteria they typically use for accepting or rejecting contributions.[15]

Consistent with the text analysis results (see Section 5.3), the survey respondents' implicit architectural thinking appears to be more congruent with a component-and-connector view than a module view of the HDFS architecture. Their daily work, after all, is more focused on the run-time behavior of the system than on concerns about the erosion of its modular structure. The survey results also suggest that there is a continuing need for architectural guidance and for refactoring. Equally importantly

[14] The Committers' questionnaire mirrored the one prepared for the Contributors since most of the Committers also contribute patches. The questions can be found in Appendix A at https://seir.sei.cmu.edu/partial/appendixa.htm .

[15] The initial email invitations were sent in January 2012and were followed by three reminders, the last of which was sent mid-February.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

many of the results also are consistent with our other results, which provides additional confidence in the overall results from a multitrait, multimethod perspective [16].

5.5.1 Results of the Analysis

Given the small number of survey respondents, all of the graphical representations in the present section are limited to counts of the number of answers selected in response to each "closed ended" survey question. The bar charts that follow in this section are arranged in a left-to-right, quasi-Pareto order from the most to the least common replies. We, of course, also included verbatim answers to the "open ended" questions that provide a useful basis for clarification and further conjecture.[16]

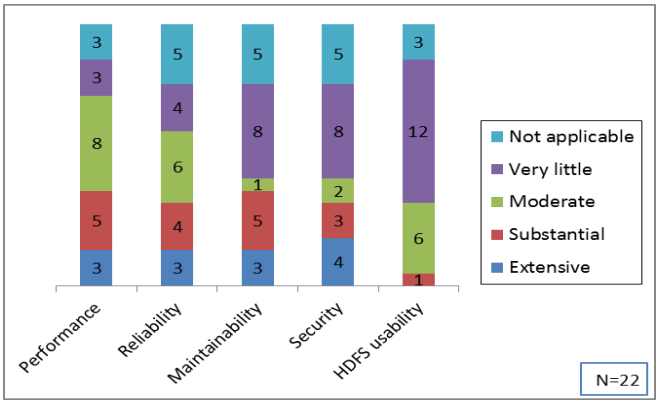

Fig. 13 Key quality attributes: Extensive or substantial errors encountered by the survey respondents

The survey results cohere well with those found in the text analysis as evidence of architectural thinking among the Contributors and Committers. The first set of questions addressed the extent to which the problems faced by HDFS are amenable to architectural solutions and recognized as such by the respondents. As seen in Figure 13, the respondents indicated that achieving high performance (e.g., response time, latency, completion time, throughput, and scalability) gave them the most difficulty in preparing their HDFS contributions. That is not surprising since performance is one of the main quality attribute concerns of HDFS. Reliability was the survey respondents' next most common concern. That too is consistent with the other main quality attribute focus that we have observed elsewhere.

Modifiability and portability were not mentioned explicitly in the questionnaire, but both are implied by "Maintainability and extensibility of existing or future system functionality, e.g., unclear modularity or module decomposition, balancing coupling and cohesion, understanding layers or other design rules, modules within a module, classes with more connections to other modules than to their own." Maintainability/extensibility was of lesser concern, which is not surprising given our observations elsewhere that a module view of the architecture was less salient to project members than a component-and-connector view. Note, however, that the Committers did report having seen major errors rather frequently with respect to maintainability as well as performance in the contributions that they reviewed. [17]

The survey respondents' concerns were focused more heavily on implementation details than architectural considerations per se. As can be seen in Figure 14, the respondents did mention having faced architecture-related difficulties in their work, particularly reducing unnecessary dependencies and propagation (e.g., identifying cyclic dependencies between classes in the source code). However, the details of defect reduction (locating bugs and determining their causes) and testing (finding test suites well-suited to the contributions and regression testing to identify infinite loops, memory leaks, data corruption or error conditions) were of greater concern to the respondents than architecture-related quality attributes. Similar results not shown here exist for the Committers' reports of having seen major errors in the contributions that they reviewed.

A similar propensity can be seen in the Committers' criteria for making their decisions about accepting or rejecting contributions. The Committers cite *all* of the criteria as being important for their decision making. Not surprisingly, however, their immediate focus rests heavily on testing. [18]

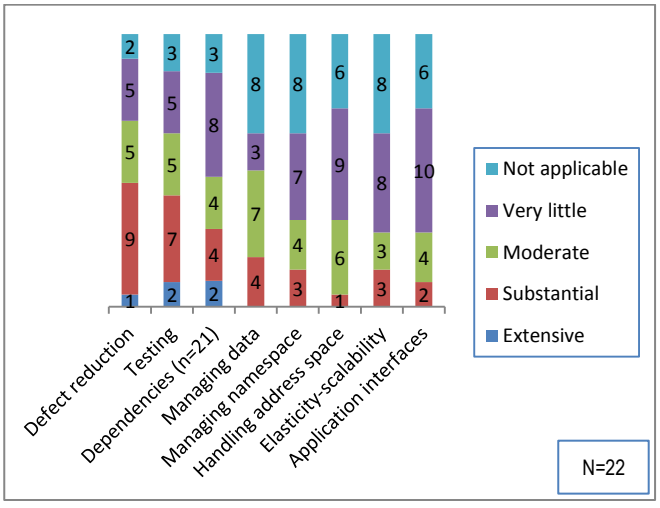

Fig. 14. Other sources of difficulties encountered in respondents' own contributions

We also included a few questions in the survey on another one of the original research questions for this study. Namely, to what extent can architectural documentation provide necessary (although not sufficient) support to recruit and retain enough Contributors and Committers to

[16] As seen in Appendix B at https://seir.sei.cmu.edu/partial/appendixb.htm, there were not many such replies, but some of them are thought-provoking.

[17] For a more detailed graphical representation, see Figure 1 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm. The number of Committers who participated in the survey is small, but this observation is worthy of further consideration.

[18] For a graphical representation, see Figure 2 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

continue maintaining and improving HDFS over time? The first question that we asked the survey respondents in this context was about how difficult it was for them to find help when they were unsure about their own code adversely affecting compatibility with existing HDFS functionality. Whatever sources they consulted, such as the code or discussions with others, few of them reported having difficulty finding help more than occasionally. [19] The survey respondents frequently said that their own experience often suffices; however most of them consult all of the sources about which we queried at least on occasion. [20] More to the point perhaps, they also report that experienced Committers have been the most helpful to them (Figure 15).

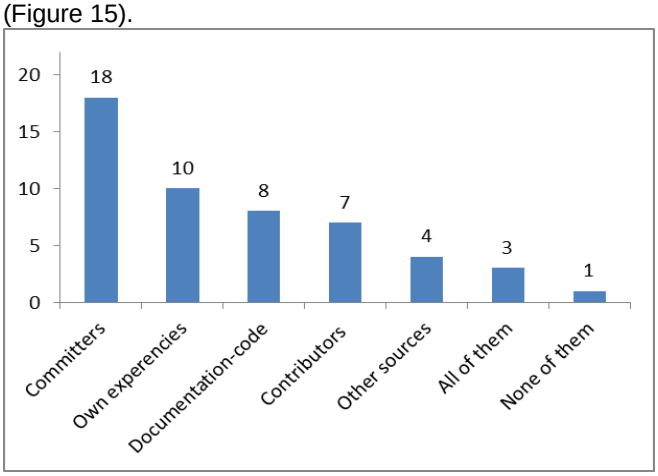

Fig. 15. Most helpful sources of guidance consulted by respondents

Finally, we asked the respondents two question sets about their familiarity with the HDFS architectural documentation that we prepared. While most of them were not initially aware of the existence of the architectural documentation,[21] a majority of the respondents had been made aware of it by one or more experienced HDFS Committers by the time of the survey.[22] Some of the respondents said that it was helpful or said they would look at it later. Others expressed skepticism about the value of it, or any other such guidance.

5.5.2 Discussion

In summary, the Contributors and Committers we surveyed clearly recognized the importance of the concepts that were covered in the architectural document and used them in their own work on the HDFS codebase. However, they sometimes tended to focus more heavily on implementation details than architectural considerations *per se* (RQ 5.1). The Committers and Contributors reported having made relatively little reference to the architectural document itself (RQ 1.1, RQ 1.2). As just mentioned in Section 5.5.2, however, the HDFS Committers appeared to be beginning to advise others about the existence of the documentation when the survey was fielded relatively soon after the architectural document was first made available on the Internet.

With the logic of hindsight, it probably would have been better to have done the text analysis prior to fielding the survey. However, short surveys focused in more depth can also clarify interpretations of text analytic and other quantitative results in a cyclic manner over time. Similarly, surveys probably are the best way to ask for the project members' judgments about the usefulness of the architecture documentation. Occasional short, focused surveys can tease out information that simply is not available elsewhere. Over the long run, that may be one of the best ways to support updating architectural guidance documentation that serves the needs of the HDFS community of Commenters, Committers and Contributors on a timely basis.

6 Concluding Observations, Recommendations, and Future Work

Finally, we would like to reflect on our journey in researching this topic. We began with a hypothesis about the effects of explicit, structured architecture documentation. This gave us a lens through which to view the HDFS project. This lens was useful in that it helped to structure our investigation and shape the research methods that we employed and the research questions that we posed. This led us to adopt a multi-method approach to understanding the study. In the course of prosecuting this research, we uncovered phenomena that challenged some of our assumptions. This forced us to look into alternative explanations for how the HDFS community maintained intellectual control over its architecture, and the effects of architectural documentation on a large, distributed community of developers.

6.1 Findings

Let us first summarize the major findings of our research:

1. The HDFS *Committers* appear to maintain intellectual control over their code base. They can do this because the code base is not very large and because they are a relatively small, stable, and co-located group. Therefore, the architecture documentation did not interest or affect the Committers (as measured by their response to our surveys and their lack of referring to it in the JIRA postings and mail archives).
2. The HDFS community of Committers (and the most active Contributors) is actively interested in architectural concepts, as shown by the text analysis that we did. They frequently discuss architectural concepts, but this architectural information is implicitly shared, and informally presented and disseminated, dispersed in mail archives and JIRA postings.
3. The architecture documentation that we created *did* appear to have an effect on the project, but principally on "outsiders," such as, less active Contributors, and

[19] See Figure 3 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm.

[20] See Figure 4 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm.

21 See Figure 5 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm.

22 See Figure 6 in Appendix C at https://seir.sei.cmu.edu/partial/appendixc.htm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

newcomers. Many people accessed the documentation web pages (over 25,000 to date) and there were many return visitors (33%, more than 8,000, of the page visits were return visits).

4. The project's social network became less centralized and the speed of promotion from Commenter to Contributor was (statistically significantly) quicker after the introduction of the documentation. Newcomers are not participating more (in terms of volume of contributions), but they are becoming more effective. This is, of course, correlation, and does not prove causation. While the lessening of centralization and increasing promotion rates are consistent with the claimed effects of explicit architectural documentation, there may be other explanations for both phenomena.
5. The architecture documentation did not appear to significantly affect submission activity or quality. There was more activity after the introduction of the architecture documentation, but this may have been due to confounding factors.

6.2 Threats to Validity

It is important at this stage to discuss any limits and threats to the validity of the findings we have presented. Threats to validity of any study can come in several forms and from different directions. For instance, internal *validity* is a critical examination of whether the experimental treatment (in our case, the introduction of the architecture documentation to the HDFS project) makes a difference; that is, whether the independent variable actually causes the changes seen in the dependent variables being examined. There are some inherent limitations in our study; for example, there was no control group, so it was unfeasible to run it as a classic controlled experiment. Since this study uses a pre/post design without a control, we can make no inference of causality, but can only observe relationships that may merit further study. Therefore, we cannot establish internal validity of the findings; those findings must be intended as observed correlations, whose causal linkage to architectural documentation still remains to be explored and validated.

External validity is a critical examination of whether the results are generalizable. We are convinced that one valuable by-product of the work we have presented is that it offers a template for carrying out analogous studies in other projects and within other contexts, which could be used to further explore the same (or similar) topics of architectural knowledge, awareness, and documentation in OSS. That could provide replication, validation, and generalization insights with respect to the findings. The key elements for replicability in this case are the choice of the OSS project for the study, which must be of (at least) moderately large scale, and, most importantly, have a community that works and communicates primarily in a distributed fashion (even in the presence, like in HDFS, of industrial involvement). Also, the software being developed should present non-trivial implications from an architectural point of view (in our case, HDFS deals with an involved distributed system, which is itself a primary component for many Cloud computing infrastructures) but the architectural knowledge should be mostly implicit within the community. There should be a way to involve (a subset of) the primary stakeholders and head project figures in the study, to both facilitate the elicitation of architectural knowledge into explicit documentation, and have a channel to endorse and publicize the documentation in front of the community. And, of course, sources of data and information about the project that are analogous (or, if possible, richer) to the ones we used should be available for a prolonged period of time.

Construct validity is established by asking whether the measures being used accurately measure what they are intended to measure. Construct validity is, therefore, a precursor to internal validity. The threats to validity that affect our study findings, and which we present below, are primarily focused on construct validity.

A major threat to validity in the raw data analysis (see Section 5.1) is a threat that is inherent in any case study: the HDFS community itself may have matured during the course of this study, creating a greater inherent interest in architectural documentation, irrespective of any activity on our part. We have no evidence supporting this threat, but it must be mentioned. Furthermore, it might appear that there is a potential threat with our analysis in terms of population validity. We cannot know who accessed the documentation, and hence we cannot say with any certainty that it was primarily accessed by Committers, Contributors, or other interested parties who had no direct connection with the project. As shown in Section 5.5, we can rule out one possibility: it appears that the Contributor and Committer groups—the ones who had the most a priori knowledge of the architecture—were unaware of the existence of our documentation. As a result, the population that we affected appeared to be the larger, mostly loosely connected group of outsiders. The HDFS committer group[23] was highly co-located: 42 of the 55 Committers are in the same time zone (GMT-8; primarily northern California) and 44 of the 55 work for just 6 companies: Yahoo!, Cloudera, Hortonworks, LinkedIn, IBM, and Facebook. Furthermore, Hortonworks was spun off from Yahoo in 2011[24], meaning that almost half of the Committer group (26 of 55) had close and recent ties to a single company. Committers, besides having deeper expertise, also work in proximity to each other and can occasionally have face-to-face meetings. Thus, the architectural documentation is not of primary interest and benefit to this community. Explicit, shared architecture documentation really benefits the project members who are not part of this core group—potential users, users, Commenters, and potential or new Contributors. This observation is consistent with one of the main benefits that has been long posited for architectural documentation [14]: it “stands in” for the architects when they are not physically present; that is, it provides insight and rationale into the most im-

[23] http://hadoop.apache.org/who.html

[24] http://www.informationweek.com/software/information-management/hortonworks-hadoop-dilemma-get-rich-givi/240007499

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

portant system-wide design decisions that affect the achievement of system qualities.

Concerning threats to construct validity related to the findings on the promotion data set, we have tried to examine the data from multiple viewpoints. That way, we were able to rule out at least one factor that could be another cause of how quickly a Commenter can be promoted to the Contributor role, i.e., the intensity of newcomers' participation. Taken together, the two statistical tests we presented in Section 5.2 are consistent with the hypothesis that the architectural documentation may have a facilitating role for newcomers to the HDFS community, who now tend to be able to contribute earlier to the code base. It is, however, more difficult, using just the data we could mine from HDFS project repositories, to eliminate other possible confounding factors, in particular, factors that have to do with how the nature of project may have evolved over time. For example, it is possible that the information contained in JIRA, the code itself, and other implicit sources, had reached a "tipping point" where many questions could be answered by a search of these sources, and this tipping point just happened to coincide with the introduction of our architectural documentation. We believe that this explanation is unlikely, as it required a substantial amount of work on our part to collect enough information to understand HDFS and create the architectural documentation in the first place, but it is a possibility we cannot positively rule out.

More generally, factors affecting project evolution could be any characteristics of the structure, process, and practices that were more significant in earlier stages of the HDFS project and which may have made it harder or slower at the time to graduate from Commenter to Contributor. We do not know of any such characteristics, but we cannot exclude such a possibility. Another set of confounding factors may descend from other changes to the project that occurred during the period of our study. In fact, we know of a few possible such incidents. For instance, we could see that, as the HDFS project was becoming more mature, new companies were engaging in it with additional development personnel. Also, during the period of our study, the project's website was improved and some additional documentation for HDFS appeared [9], unconnected with our own architectural documentation effort. Those changes may, of course, have contributed to the observed phenomenon of quicker promotion from the Commenter to the Contributor role, but we had no chance to assess whether those changes had any statistical effects, or to measure their magnitude.

Another threat to validity is the "history effect." A number of uncontrolled events occurred that may have confounded our observations. First, HDFS acquired a number of additional users during the period of study, and this could have led to an influx of new Contributors and Commenters irrespective of the architectural document. Second, the new users might have produced poorer quality code (more likely to lead to a rejected patch) because of inexperience. Third, these additional users might have led to a change in demand (either in volume or in how HDFS was used), thus confounding counts of issues and patches. Fourth, as the project matured, the difficulty of problems may have changed. Newer projects may address more difficult problems first, leaving lesser problems for later. On the other hand, over time the code base becomes increasingly large and complex. We did not attempt to evaluate the difficulty of the issues addressed over time. Fifth, the Committers' standards might have changed over time as they gained experience.

The fact that we had consistent results across distinct, complementary methods is our response to these threats to validity. Multi-method exploratory data analysis of the kind we are doing is often necessary, especially in the early stages of research and where the body of empirical research is slender. Our work is a single study in an as yet small body of empirical research. Such research *needs* to be exploratory and multi-method.

6.3 Implications for Architecture Documentation

Architecture documentation may be implicit or explicit. These corpora serve as implicit mediators among the project participants, and serve as carriers of project knowledge. The HDFS architecture community—the Committers in particular—had evolved its own (largely informal) mechanisms for documenting and sharing architectural knowledge. Since no individual was *the* acknowledged architect, no one had the duty to maintain a representation of the architecture. Some explicit representations of architectural knowledge did emerge, but the vast majority of this knowledge remained implicit and difficult to access for newcomers.

We found that this "emergent" architecture, as documented in the HDFS Contributors' informal text interchanges, was mostly adequate and reflected the most important project concerns. The project's approach to architecture may not, however, scale up. An analysis of the project's module structure shows that there were already some poor modularity practices, such as cyclic dependencies, that will make the project harder to maintain and evolve over time [74].

Architecture documentation serves as a kind of "information radiator" in that it supports the broadcast of architectural information. Human information networks are constrained by latency and individual bandwidth. For example, a single highly connected person may be followed by a huge number of people but can only respond to a limited number of questions. This view is consistent with one of the main benefits that has been long posited for architectural documentation [14]: it "stands in" for the architects when they are not physically present; that is, it provides insight and rationale into the most important system-wide design decisions that affect the achievement of system qualities.

6.4 Recommendations

Based on our findings, we can provide some recommendations for projects that grow beyond a point where smart, motivated programmers can manage the complexity simply via textual archives and casual transmission of project knowledge. Start with textual analysis of project communications to identify important concepts and con-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2465387, IEEE Transactions on Software Engineering

cerns [29]. Then reverse engineer the code base to identify architectural dependencies and to establish connections between architectural concepts and code modules. In this way, the architecture can be made explicit with little manual effort and important architectural knowledge can be captured [21], [36].

Equally important, much attention should be paid to improving the user interface of the online documentation to support flexible browsing through multiple views of the architecture. And the documentation must be kept up to date to remain useful to the wider community. The Hadoop community has, to a certain extent, already been learning this lesson. They experimented with a "heatmap" view of terms representing key software elements in their JIRA archive, and have for a long time organized their issue data on the Hadoop website according to their view of software architectural components. These additions to the website interface seem like responses to a concern with making it easier for users to find relevant information on important software development issues.[25]

Our results have other implications that may greatly improve the access to timely architectural guidance for the HDFS community and for other similar projects. The same text analytic methods that we used for this research can be used to provide detailed guidance for immediate problem-solving purposes, which also may encourage further, explicit consideration of architectural principles. Text analysis methods can be used to provide free-form queries with links to specific JIRA discussion threads and the code itself to help Contributors and Committers consider previous solutions to similar problems. With the right user interface, the same methods can be unobtrusively indexed by both implicit architectural concepts and updated HDFS architectural guidance to support flexible browsing through multiple views of HDFS architectural guidance documentation.

In addition to our specific recommendation about architecture documentation practice, we have come to believe that the research approach that we evolved, over the course of our three-year study into HDFS, has useful lessons for other software engineering research. In particular, we believe that the multitrait, multimethod approach to the research that we adopted can be a model for other software engineering research where controlled studies are impossible, and where there are many possible confounding factors that complicate the analysis of project activities and characteristics in any single study [35], [38].

In small-scale projects, software development teams of five plus or minus two were able to share a "theory" of the program they were developing through oral discussions with a little or no documentation of any kind [49]. In fact, Naur argued that in subsequent sustainment of the program, explicit documentation of its theory was no substitute for the involvement of one or another of the original programmers who created and shared the theory when the program was initially developed.

For larger projects, Scacchi (and others) have been arguing that ad hoc documentation not intended as an explicit rendering of requirements can be used to describe, proscribe, or prescribe what's happening in an OSS project [59]. Implicit, ad hoc documentation as communication and messages within project email, threaded message discussion forums, bulletin boards, group blogs, news postings, instant messaging, project digests, how-to guides, to-do lists, frequently asked questions, project wikis, and external publications can all be used in place of explicit software engineering documentation [59]. Such informal documentation seems to be working so far for the HDFS project. This may be one of the reasons for the next finding.

Studies have shown that as projects grow and reach a certain size, modularization is crucial in allocating work [6]. If the HDFS project grows beyond a point where smart programmers can manage the complexity simply via casual transmission of project knowledge, text analysis and the other techniques employed in this study could be useful. Text analysis could identify the important architectural concepts, what views they instantiate, and where further concepts are needed to complete the views or introduce views that are absent. In the case of HDFS, it has already shown where the module view is incomplete and how it could be combined with an "allocation of work" view that is currently missing.

Architecture is, in the end, a bunch of shared conventions. We believe that the larger the project the more that these conventions need to be made explicit and structured—that is to say, the more that a project benefits from structured, explicit architecture documentation. We noted in our introduction that architecture documentation serves three main purposes: education, construction, and analysis. The HDFS project's ad hoc, emergent architecture documentation appeared to fill much of the documentation needs of the project as it grew, but this was not without risks. Several of the Committers were worried about the future of the project, particularly its ability to attract and train new Contributors and Committers.[26] The HDFS project has clear modularity risks and these likely will not be addressed by the emergent architecture and the social processes that shape it.

6.5 Future Work

We see a number of avenues for addressing the threats to validity identified earlier. More thorough analysis of the data prior to beginning the intervention could have verified stability of the data. More careful timing of the introduction of the architectural documentation may have avoided confounding factors or helped select better pre- and post-matching periods. Of course, such analysis might also guide a researcher to choose an alternative subject for intervention. In addition, considering a longer time-scale may reveal effects that were not observable in the modest period of this study.

[25] https://issues.apache.org/jira/browse/HADOOP#selectedTab=com.atlassian.jira.plugin.system.project%3Alabels-heatmap-panel

[26] Personal communication.

Partly as a result of the research we did for this paper, we now are working on deriving quantitative measures based on text analytic results. The categorizations and relationships uncovered using text analytic methods are of course found by applying statistical methods in the free-form textual source documents. Similar work has been done on a smaller scale for years in analyzing answers to open-ended questions in survey research. Those derived measures can then be joined and analyzed together with quantitative measures from other sources.

We believe that architecture needs to be made explicit to avoid eventual project erosion. Our team, in consultation with the HDFS leadership, should consider modifying the existing architectural documentation to focus more explicitly on distinctions between a component-and-connector view and a module view of the architecture. Similarly, architecture-related testing and review requirements could be made more explicit, e.g., no check-ins that violate modularization rules. We are investigating these sorts of interventions that could lead to *sustainable* architecture documentation practices.

ACKNOWLEDGEMENTS

We thank Len Bass and Ipek Ozkaya, for their work in helping to document the HDFS architecture. We also thank Forrest Shull and Dave Zubrow for their insightful comments that helped to shape our arguments.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0001095

REFERENCES

[1] T. Alspaugh and Walt Scacchi, "Ongoing software development without classical requirements", In *21st IEEE International Requirements Engineering Conference (RE'13)*, pp. 165–174, July 2013.

[2] J.M. Anthonisse, *The Rush in a Graph*. Amsterdam: Mathematische Centrum, 1971.

[3] Apache Hadoop, "*HDFS Architecture,*" http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html (March 2014).

[4] Atos Origin, "Method for Qualification and Selection of Open Source Software (QSOS) Version 1.6," April 2006, http://www.qsos.org.

[5] M.A. Babar et al., *Software Architecture Knowledge Management: Theory and Practice,* Springer, 2009.

[6] C. Baldwin and K. Clark, *Design Rules, Vol 1: The Power of Modularity*, MIT Press, 2000.

[7] L. Bass, R. Kazman, and I. Ozkaya, "Developing Architectural Documentation for the Hadoop Distributed File System," *Open Source Systems: Grounding Research, Proceedings of the 7th International Conference on Open Source Systems (OSS2011),* Springer, 2011, pp. 50-61.

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 3rd ed.,* Addison-Wesley, 2013.

[9] A. Brown and G. Wilson (eds.), *The Architecture of Open Source Applications*, lulu.com, 2012.

[10] M. Callon, J. Law, and A. Rip (eds.), *Mapping the Dynamics of Science: Sociology in the Real World*. London: Macmillan Press, Ltd., 1986.

[11] A. Cambrosio et al., "Historical Scientometrics? Mapping Over 70 Years of Biological Safety Research with Co-Word Analysis," *Scientometrics*, vol. 27, no. 2, 1993, pp. 119-143.

[12] A. Cambrosio et al., "Big Data and the Collective Turn in Biomedicine: How Should We Analyze Post-genomic Practices?" *TECNOSCIENZA Italian Journal of Science & Technology Studies* vol. 5, no. 1, 2014, pp. 11-42.

[13] D. Campbell and J. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Houghton Mifflin, 1963.

[14] P. Clements et al., *Documenting Software Architectures: Views and Beyond, 1st ed.,* Addison-Wesley, 2002.

[15] N. Coulter, I. Monarch, and S. Konda, "Software Engineering As Seen Through Its Research Literature: A Study in Co-Word Analysis," *Journal of the American Society for Information Science*, vol. 49, no. 13, 1998, pp. 1206-1223.

[16] J.W. Creswell and V.L.P. Clark, "Designing and Conducting Mixed Methods Research." In *Designing and Conducting Mixed Methods Research*. Sage, 2007.

[17] K. Crowston and J. Howison, "The Social Structure of Free and Open Source Software Development," *First Monday. Vol.* 10, 2005, pp. 2-7.

[18] K. Crowston, J. Howison, and H. Annabi. "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process Improvement and Practice*, vol. 11, no. 2, March-April 2006, pp. 123-148.

[19] K. Crowston et al., "Free/Libre Open-Source Software Development: What We Know and What We Do Not Know," *ACM Computing Surveys*, vol. 44, no. 2, 2012.

[20] J-M Dalle and P. David, "'It Takes All Kinds': A Simulation Modelling Perspective on Motivation and Coordination in Libre Software Development Projects." Stanford Institute for Economic Policy Research, SIEPR Discussion Paper, no. 07-24, December, 2007.

[21] Wei Ding et al., "Knowledge-Based Approaches in Software Documentation: A Systematic Literature Review," *Information and Software Technology,* vol. 56, no. 6, June 2014, 545–567

[22] Wei Ding et al., "How Do Open Source Communities Document Software Architecture: An Exploratory Survey," *Proceedings of the 19th International Conference on Engineering of Complex Computer Systems (ICECCS)*, IEEE Computer Society, 2014.

[23] K. Ehrlich and M. Cataldo, "All-for-One and One-for-All?: A Multi-Level Analysis of Communication Patterns and Individual Performance in Geographically Distributed Software Development." *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW 2012),* February 2012.

[24] R. English and C. Schweik, "Identifying Success and Tragedy of FLOSS Commons: A Preliminary Classification of Sourceforge. net Projects," *Proceedings of the 1st International Workshop on Emerging Trends in FLOSS Research and Development*, IEEE Computer Society, May 2007.

[25] L.C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, 1977, pp. 35-41.

[26] P. Giuri, F. Rullani, and S. Torrisi, "Explaining Leadership in Virtual Teams: The Case of Open Source Software," *Infor-*

mation Economics and Policies, vol. 20, no. 4, December 2008, pp. 305-315.

[27] S. Ghemawat, H. Gobioff, and S-T Leung, "The Google File System." *ACM SIGOPS Operating Systems Review* vol. 37, no. 5, December 2003, 23-43.

[28] Bernard Golden, "Making Open Source Ready for the Enterprise: The Open Source Maturity Model," May 2008, http://timreview.ca/article/145.

[29] A. Guzzi et al., "Communication in Open Source Software Development Mailing Lists," *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, California, pp. 2013, 277-286.

[30] M.A.K. Halliday and J.R. Martin, *Writing Science: Literacy and Discursive Power*, 1st ed., Taylor & Francis, June, 1993.

[31] Z. Harris et al., *The Form of Information in Science: Analysis of an Immunology Sublanguage*. Boston Studies in the Philosophy of, Science, 104, Kluwer Academic, 1989.

[32] P. He, B. Li, and Y. Huang, "Applying Centrality Measures to the Behavior Analysis of Developers in Open Source Software Community," *Proceedings of The Second International Conference on Cloud and Green Computing (CGC 2012)*, November 2012.

[33] C. Holt, "Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages," *International Journal of Forecasting*, vol. 20, no. 1, January-March 2004, pp. 5–10.

[34] R. Hyndman and Y. Khandakar, "Automatic Time Series Forecasting: The Forecast Package for R," *Journal of Statistical Software* vol. 27, no. 3, 2008.

[35] Abraham, Kaplan, *The Conduct of Inquiry: Methodology for Behavioral Science*. Chandler Publishing Company, 1964.

[36] R. Kazman, S. J. Carriere, "Playing Detective: Reconstructing Software Architecture from Available Evi-dence", *Automated Software Engineering*, April 1999, 6:2, 107-138.

[37] R. Kazman and H-M Chen, "The Metropolis Model: A New Logic for the Development of Crowdsourced Systems," *Communications of the ACM*, July 2009, pp. 76-84.

[38] Thomas S. Kuhn, *The Structure of Scientific Revolutions: 50th Anniversary Edition*, University of Chicago Press, 2012.

[39] Philippe Kruchten, "The 4 + 1 View Model of Architecture," *IEEE Software,* vol. 12, no. 6, 1995, pp. 42-50.

[40] Lattix. http://www.lattix.com (retrieved Mar. 3, 2015).

[41] J. Lave and E. Wenger, *Situated Learning: Legitimate Peripheral Participation,* Cambridge University Press, 1991.

[42] S-Y. T Lee, H.-W. Kim, and S. Gupta, "Measuring Open Source Software Success," *Omega* vol. 37, no. 2, 2009, pp. 426-438.

[43] Y. Long and K. Siau, "Social Network Structures in Open Source Software Development Teams," *Journal of Database Management*, vol.18, no. 2, 2007, pp. 25-40.

[44] A. MacCormack, J. Rusnak, and C.Y. Baldwin, "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code." *Management Science*, vol. 52, no. 7, 2006, pp. 1015-1030.

[45] T.W. Malone et al., "Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination," *ACM Transactions on Office Information Systems*, vol. 5, no. 2, April 1987, pp. 115–131.

[46] I. Malavolta et al., "What Industry Needs from Architectural Languages: A Survey," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, 2013, pp. 869-890.

[47] A. Mockus, R.T. Fielding, and J.D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology,* vol. 11, no. 3, July 2002, pp. 309–346.

[48] I. Monarch, D. Goldenson, and L. Osiecki, "Requirements and Their Impact Downstream: Improving Causal Analysis Processes Through Measurement and Analysis of Textual Information," Technical Report CMU/SEI-2008-TR-018, Software Engineering Institute, Pittsburgh, PA], September 2008.

[49] P. Naur, "Programming as Theory Building,," *Microprocessing and Microprogramming*, vol. 15, no. 5, May 1985, pp. 253-261.

[50] "Business Readiness Rating for Open Source—A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software," BRR 2005 - RFC 1.

[51] C. Osterlund & K.K. Crowston, "Boundary-Spanning Documents in Online FLOSS Communities: Does One Size Fit All?" *Proceedings of 46th Hawaii International Conference on Systems Sciences (HICSS-46)*, pp. 1600 – 1609, Jan. 2013.

[52] D. Pagano and W. Maalej, "How Do Open Source Communities Blog?" *Empirical Software Engineering*, vol. 18 no. 6, 2013, pp. 1090-1124.

[53] E. Petrinja and G. Succi, "Assessing the Open Source Development Processes Using OMM." *Advances in Software Engineering*, vol. 8, 2012.

[54] D. Rost et al., "Software Architecture Documentation for Developers: A Survey," *Proceedings of the 7th European Conference on Software Architecture (ECSA),* pp. 72-88, 2013.

[55] P. Runeson et al., *Case Study Research In Software Engineering: Guidelines and Examples* (1st ed.). John Wiley & Sons, Inc, 2012.

[56] Warren Sack et al., "A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software." *Computer Supported Cooperative Work (CSCW)*, vol. 15, issue 2-3, June 2006, pp. 229-250.

[57] B. Sadowski, G. Sadowski-Rasters, and G. Duysters, "Transition of Governance in a Mature Open Source Software Community: Evidence from the Debian Case," *Information Economics and Policies*, vol. 20, no. 4, December 2008, pp. 323-332.

[58] W. Scacchi, "Understanding the Requirements for Developing Open Source Software Systems," *IEEE Proceedings on Software*, vol. 149, no. 1, 2002, pp. 24 -39.

[59] W. Scacchi, "Understanding Requirements for Open Source Software," *Design Requirements Workshop, LNBIP 14*, pp. 467–494, 2009.

[60] K. Shvachko et al., "The Hadoop Distributed File System," *IEEE 26th Symposium on Mass Storage Systems and Technologies*, pp. 1-10, 2010.

[61] E. Smith and M.S. Humphreys, "Evaluation of Unsupervised Semantic Mapping of Natural Language with Leximancer Concept Mapping," *Behavior Research Methods*, vol. 38, no. 2, 2006, pp. 262-279.

[62] SonarJ. http://www.hello2morrow.com/products/sonarj (Retrieved June 015)

[63] K. Stephenson and M. Zelen, "Rethinking Centrality: Methods and Applications", *Social Networks,* vol. 11, 1989, pp. 1-37.

[64] K. Stol et al., "The Use of Empirical Methods in Open Source Software Research: Facts, Trends and Future Directions," *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*," pp. 19-24, 2009.

[65] C. Subramaniam, R. Sen, and M.L. Nelson, "Determinants of Open Source Software Project Success: A Longitudinal Study," *Decision Support Systems* vol. 46, no. 2, 2009, pp. 576-585.

[66] D. Taibi, L. Lavazza, and S. Morasca, "OpenBQR: A Framework for the Assessment of OSS." *Open Source Development, Adoption and Innovation*, 2007, pp. 173-186.

[67] R. Taylor, N. Medvidovic, and E. Dashovy, *Software Architecture: Foundations, Theory, and Practice*, Wiley, 2009.

[68] H. Unphon and Y. Dittrich, "Software Architecture Awareness in Long-Term Software Product Evolution," *Journal of Systems and Software*, Elsevier, vol. 83, no. 11, 2010, pp. 2211-2226.

[69] J.S. van der Ven and J. Bosch, Making the Right Decision: Supporting Architects with Design Decision Data," *Proceedings of the 7th European Conference on Software Architecture (ECSA),* pp. 176-183, 2013.

[70] X. Wang et al., "Microblogging in Open Source Software Development: The Case of Drupal Using Twitter," *IEEE Software*, 2013.

[71] A. Wiggins and K. Crowston, "Reclassifying Success and Tragedy in FLOSS Projects." *Open Source Software: New Horizons*. Springer Berlin Heidelberg, pp. 294-307, 2010.

[72] T. Wolf, A. Schroter, and D. Damian, "Mining Task-Based Social Networks to Explore Collaboration in Software Teams," *IEEE Software*, vol. 26, no. 1, January-February 2009, pp. 58-66..

[73] S. Wong et al., "Detecting Software Modularity Violations," *Proceedings of ICSE 2011*, pp. 411-420, 2011.

[74] L. Xiao, Y. Cai, and R. Kazman, "Design Rule Spaces: A New Form of Architecture Insight," *Proceedings of the International Conference on Software Engineering (ICSE) 2014*, June 2014.

[75] R.K. Yin, *Case Study Research: Design and Method*s, 5th edition. Sage, 2009.

[76] M.S. Zanetti, "The Co-Evolution of Socio-Technical Structures in Sustainable Software Development: Lessons from the Open Source Software Communities," *Proceedings of the 34th International Conference on Software Engineering (ICSE 2012),* May 2012.

[77] M.S. Zanetti et al., "Categorizing Bugs with Social Networks: A Case Study on Four Open Source Software Communities," *Proceedings of the International Conference on Software Engineering*, pp. 1032-1041, 2013.

Rick Kazman is a professor at the University of Hawaii and a principal researcher at the Software Engineering Institute (SEI) of Carnegie Mellon University. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. Kazman has created several highly influential methods and tools for architecture analysis, including the SAAM (Software Architecture Analysis Method), the ATAM (Architecture Tradeoff Analysis Method), the CBAM (Cost-Benefit Analysis Method) and the Dali and Titan tools. He is the author of over 150 peer-reviewed papers, and co-author of several books, including *Software Architecture in Practice*, *Evaluating Software Architectures: Methods and Case Studies*, and *Ultra-Large-Scale Systems: The Software Challenge of the Future*.

Dennis Goldenson joined the SEI in 1990 after teaching at Carnegie Mellon University since 1982. A senior member of both IEEE and ACM, his work has focused on modeling performance and quality outcomes of software-reliant systems. Recent work in addition to empirical analysis of software architecture includes calibration of expert judgment in the presence of uncertainty, early cost estimation, systems engineering effectiveness, statistical methods to ensure data quality, and requirements engineering. Goldenson's related interests are in tools to support collaborative processes, the quantitative analysis of textual information, experimental design, survey research methods, and the visual display of quantitative information.

Ira Monarch has investigated process and design issues in large-scale engineering programs, both military and industrial, for over twenty years. At the Software Engineering Institute, he led and participated in projects that developed and used text analytic tools for uncovering patterns and identifying risks and failure conditions in software design, architecture, development, and maintenance. Additionally, he led a project to explore the feasibility of using communities of practice, knowledge sharing networks, and interactive knowledge bases to support the transition and adoption of software engineering technologies. Mr. Monarch was awarded am SEI independent research and development grant to develop tools and techniques to investigate why large-scale software projects fail at alarmingly high rates. Prior to his work at the SEI, he participated in computational linguistics projects at Carnegie Mellon University where he led work to develop knowledge-based thesauri to support machine translation and to improve noun phrase parsing in information retrieval.

William Nichols is a senior member of the IEEE and a senior member of technical staff at the Software Engineering Institute at Carnegie Mellon University. He has more than 25 years of technical and management experience in the software engineering industry. His current work focuses on software process measurement, project management, quality, security, and improving development team performance. During his tenure at the SEI, Dr. Nichols has worked extensively with the Team Software Process (TSP) Initiative, where he currently serves as a Personal Software Process (PSP) instructor and a TSP Mentor Coach. Prior to joining the SEI, Nichols earned a doctorate in physics from Carnegie Mellon University after completing graduate work in High Energy Physics. He later led a software development team at the Bettis Laboratory near Pittsburgh, Pennsylvania, developing nuclear engineering and scientific software.

Giuseppe Valetto received a Laurea degree in Electronic Engineering from Politecnico di Torino, Italy in 1992, an MS in Computer Science from Columbia University, USA in 1994, and a Ph.D. in Computer Science, again from Columbia University, in 2004. Valetto has mostly worked on collaborative software engineering and distributed and self-adaptive systems. He has held positions as Xerox Research, CEFRIEL-Politecnico di Milano, Telecom Italia Lab, IBM T.J. Watson Research Center, and Drexel University. He is currently a researcher in the Distributed Adaptive Systems research unit at Fondazione Bruno Kessler, Italy. He is a member of IEEE.