Information & Management xxx (2016) xxx–xxx

Contents lists available at ScienceDirect

Information & Management

journal homepage: www.elsevier.com/locate/im

Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution

Sherlock A. Licorish*, Stephen G. MacDonell

Department of Information Science, University of Otago, PO Box 56, Dunedin 9054, New Zealand

ARTICLE INFO

Article history:
Received 11 November 2014
Received in revised form 21 June 2016
Accepted 22 September 2016
Available online xxx

Keywords:
Software developers
Task differences
Participation and engagement
Speed of task resolution
Mining software repositories
Empirical studies

ABSTRACT

In seeking to understand the processes enacted during software development, an increasing number of studies have mined software repositories. In particular, studies have endeavored to show how teams resolve software defects. Although much of this work has been useful, we contend that large-scale examinations across *the range of* activities that are commonly performed, beyond defect-related issues alone, would help us to more fully understand the reasons why defects occur as well as their consequences. More generally, these explorations would reveal how team processes occur during all software development efforts. We thus extend such studies by investigating how software practitioners work while undertaking the range of software tasks that are typically performed. Multiple forms of analyses of a longitudinal case study reveal that software practitioners were mostly involved in fixing defects, and that their engagement covaried depending on the nature of the work they were performing. Furthermore, multiple external factors affected speed of task resolution. Our outcomes suggest that behavioral and intrinsic issues may interact with extrinsic factors becoming significant predictors of the speed of software task resolution.

1. Introduction

Software development is generally acknowledged as an intellectually challenging activity, and one that typically requires team members to work collectively to create a product or service that may be critical but yet is conceptual, fluid, and intangible. In spite of the ongoing provision of innovations with respect to development methods and tools [1–3], research continues to note that many software projects do not succeed [4–6]. In recent years, an increasing proportion of research effort has therefore aimed to understand the work practices and team processes implemented during development, in the belief that this knowledge might be better leveraged to improve project outcomes [7,8]. In particular, automatically generated archives and repositories have gained prominence as sources of information for those studying team behaviors, enabling researchers to study software practitioners' involvement in detail, and performance in development and maintenance activities. Such studies have examined both open and closed repositories of projects including Apache [9], Eclipse [10], GNOME, NetBeans, and OpenOffice [11], along with Jazz [12–15] and Windows Vista [16], providing explanations for various team phenomena and development issues and outcomes.

This growing attention given to mining software repositories reflects the emerging significance of the study of automatically stored software artifacts in order to understand team processes. More specifically, communication artifacts such as electronic messages, change request histories, and blogs are able to provide unique perspectives on the activities that occur during the software development life cycle (SDLC) – perspectives that cannot be drawn from code or similar technical artifacts. Analysis opportunities presented by these automatically recorded artifacts are also potentially valuable because of the reduction of the likely bias that can arise with self-reporting [17,18], as well as the unobtrusive nature of the investigation of team processes and work practices from such artifacts.

Although previous work has provided useful insights into the nature of the relationship between the frequency of practitioners' communications and the prevalence of bugs (or software defects) [10,19], comparatively less attention has been given to the ways in which developers work and behave when undertaking other software development tasks that are commonly performed [13,20]. In addition, although previous work has examined how communication patterns relate to team size [21,22], there is growing support for the need to examine *the details within* developers'

* Corresponding author.
E-mail addresses: sherlock.licorish@otago.ac.nz (S.A. Licorish), stephen.macdonell@otago.ac.nz (S.G. MacDonell).

http://dx.doi.org/10.1016/j.im.2016.09.005
0378-7206/© 2016 Elsevier B.V. All rights reserved.

communications beyond assessments based only on message exchange frequency [23,24]. This need to move beyond frequency-based assessments and to examine a larger spread of software tasks is supported by outcomes from early works in the social and organizational psychology space, which established that multiple properties of team tasks affect team performance [25–27]. Thus, different *task ecosystems* are likely to benefit from particular starting configurations and team arrangements. Exploring the way developers work across *the range of* software tasks that are commonly performed, and particularly those initial events that lead to the development of software features in the first place, and then subsequent bugs, could provide added value for the software development community.

We set out to address this research opportunity, and hence provide initial insights into the way software tasks are distributed in a large software project, and how teams assemble and undertake different forms of software development activity. This study thus provides an extension to those mentioned above, with a view to explaining how different task ecosystems are likely to benefit from particular team configurations. Our contributions in this paper are twofold: 1. We extract and mine a large software repository and apply both statistical and deeper content analysis (CA) approaches in our study of software artifacts to understand developers' work practices when performing various forms of software task. Our experiences are systematically documented, and may serve as a guide for those undertaking similar studies involving multiple forms of analyses. 2. Subsequently, given our observations, we decompose our findings through a range of theoretical lenses, provide multiple recommendations for software project governance, and identify avenues for future work.

The remainder of this paper is organized as follows. In the next section (Section 2), we survey previous work, and outline our specific research questions (RQs). We then describe our research model, and further decompose our RQs in Section 3. We explain our research method and measures in Section 4, also outlining our study context in this section. In Section 5, we present our results, and these findings are discussed in Section 6. In Section 7, we highlight the implications of our findings, and outline future research directions. In Section 8, we consider our study's limitations, and finally, we draw conclusions in Section 9.

2. Background and motivation

We survey previous work in this section. In order to both ground and structure our literature inspection, we first review those studies that have considered software teams' communication in Section 2.1. Considering our objective to examine how practitioners work across a range of software development tasks, we next examine general works on task differences in Section 2.2. We finally survey works that have considered software development task differences in Section 2.3, identifying research gaps and outlining our RQs in this subsection.

2.1. The study of software teams' communication

Beyond using and interpreting measurements focused on code and numbers of bugs [28,29], the availability of publicly searchable communication artifacts has provided researchers with the opportunity to study patterns in software development in far greater detail than would be possible if they were to consider technical artifacts alone [10,21]. This form of evidence has supported analysis at individual and team levels, providing insights in relation to participants' contributions to code. Such outcomes are evident in the literature on the study of communication and coordination from both open-source software (OSS) and closed-source software (CSS) repositories. In the OSS context, for instance, Abreu and Premraj [10] examined the Eclipse mailing list and found that developers communicated most frequently at release and integration time, but that increased communication also coincided with a higher number of bugs being introduced. Bird et al. [9] confirmed that the more software development an individual undertakes, the more coordination and controlling activities (s) he must perform. In considering patterns of contribution, Cataldo et al. [21] found that the practitioners who communicated the most also contributed most actively during software development. Furthermore, in a later study, Shihab et al. [30] found that proposals and actions discussed during team communication correlated with subsequent software development actions that were enacted, when studying the GNOME OSS project.

Earlier work by Howison et al. [22] found that a few key members of smaller OSS projects occupied the center of their teams' communication networks, in contrast to larger teams whose communication networks appeared more modular. Hinds and McGrath [31] confirmed the centralized communication pattern, but did not consider team size. Bird et al. [32] examined communities and subcommunities among the Apache, Python, PostgreSQL, and Perl projects, and concluded that specific technical needs drove the emergence of subcommunities in these projects. The Debian mailing list was interrogated by Sowe et al. [33] to observe knowledge-sharing among developers; they found that no specific individual dominated knowledge-sharing activities. A study of coordination conducted by Ehrlich et al. [34] found that brokers bridge communication gaps for teams that communicated across distributed sites. In addition, Ghapanchi [35] found that practitioners' willingness to communicate in OSS projects was positively influenced by task identification, and the popularity of such projects is positively influenced by who gets assigned tasks, and how such tasks are managed. These findings indeed support the view that evidence drawn from communication and coordination processes could reliably complement and support code-centric analyses, thus highlighting the importance of studying communication artifacts.

Beyond those works using OSS repository data to investigate team processes, artifacts in the IBM Rational Jazz CSS repository have also been used to study software practitioners' interactions and communications, largely from a social network analysis (SNA) perspective [36–38], offering somewhat contradictory findings to those noted in the OSS body of work. Contrary to the findings reported by others, which showed that a few developers generally dominate team communication [30,39], Nguyen et al. [37] studied multiple software teams at IBM Rational and noted that a high proportion – approximately 75% – of IBM Rational Jazz's team members actively participated in the project's communication network. In addition, these authors found IBM Rational Jazz project teams to have highly interconnected social networks, requiring few brokers to bridge communication gaps. When examining Jazz teams, and others developing software for the automotive industry, Ehrlich and Cataldo [40] found that there were improvements in teams' productivity and product quality when technical leaders shared more information, or when they occupied central positions in communication networks.

The derivation of insights such as those just described further supports the relevance of studying teams' communications. In fact, Datta et al.'s [41] SNA study of agile developers' collaboration while using the IBM Rational Jazz platform found that developers' expressions during regular communication possessed a wealth of useful information, much more than could be gleaned from examining source file changes alone. An earlier study (published in 1994) exploring software developers' activities found that up to 50% of practitioners' time was spent on interpersonal communication and coordination during software problem solving [42].