The Journal of Systems and Software 115 (2016) 82–101

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Towards semi-automated assignment of software change requests

Yguaratã Cerqueira Cavalcanti[a,*], Ivan do Carmo Machado[b], Paulo Anselmo da Motal S. Neto[c], Eduardo Santana de Almeida[b]

[a] Brazilian Federal Data Processing Service (SERPRO), Florianópols, Brazil
[b] Computer Science Department, Federal University of Bahia, Salvador, Brazil
[c] Center for Informatics, Federal University of Pernambuco, Recife, Brazil

ARTICLE INFO

Article history:
Received 29 April 2015
Revised 18 December 2015
Accepted 25 January 2016
Available online 9 February 2016

Keywords:
Software maintenance and evolution
Change request management
Automatic change request assignment
Bug triage

ABSTRACT

Change Requests (CRs) are key elements to software maintenance and evolution. Finding the appropriate developer to a CR is crucial for obtaining the lowest, economically feasible, fixing time. Nevertheless, assigning CRs is a labor-intensive and time consuming task. In this paper, we report on a questionnaire-based survey with practitioners to understand the characteristics of CR assignment, and on a semi-automated approach for CR assignment which combines rule-based and machine learning techniques. In accordance with the results of the survey, the proposed approach emphasizes the use of contextual information, essential to effective assignments, and puts the development team in control of the assignment rules, toward making its adoption easier. The assignment rules can be either extracted from the assignment history or created from scratch. An empirical validation was performed through an offline experiment with CRs from a large software project. The results pointed out that the approach is up to 46.5% more accurate than other approaches which relying solely on machine learning techniques. This indicates that a rule-based approach is a viable and simple method to leverage CR assignments.

1. Introduction

Change Request (CR) are software artifacts that describe defects to be fixed or enhancements to be implemented in a software system (Cavalcanti et al., 2013a). CRs are managed with the support of a CR repository software, such as Bugzilla (Bugzilla, 2013) and Mantis (Mantis Bug Tracker, 2013). These repositories play a fundamental role in the software maintenance process, being a common place for communication and coordination among different stakeholders (Bertram et al., 2010). Indeed, the CR artifact is the primary unit of work in many software development projects (Anvik and Murphy, 2007).

The task of assigning a CR, also known as *CR triage*, consists of selecting the most suitable software developer to handle a given CR. Generally, such a developer is the one who has enough expertise to handle the issues reported in the CR (Aljarah et al., 2011). In addition, the assignment decision must take into account the developer's workload, availability, and the CR priority, in order to obtain the lowest, economically feasible time to fix (Di Lucca et al., 2002; Hosseini et al., 2012; Cavalcanti et al., 2013c). Thus, this task requires considerable knowledge of the project, and good communication skills to negotiate with the involved stakeholders (Cavalcanti et al., 2013c).

Assigning CRs to developers is both labor-intensive and time consuming, as it is usually regarded as a manual handling task (Anvik et al., 2006; Jeong et al., 2009). Depending on the software project, the number of new CRs can vary from dozens to hundreds in a single day (Cavalcanti et al., 2013a). As a consequence, the greater the number of CRs that are opened, the more complex the problem becomes.

Several automated approaches have been proposed to overcome the problem of CR assignment by using machine learning techniques. Some of these approaches are based on the hypothesis that the most suitable developer for a new CR is the one who has already solved similar CRs in the past (Di Lucca et al., 2002; Cubranic and Murphy, 2004; Anvik et al., 2006; Ahsan et al., 2009b; Jeong et al., 2009; Lin et al., 2009; Rahman et al., 2009). Other approaches consider that an appropriate developer can be found by looking at past CRs and data from version control systems (Canfora and Cerulo, 2006; Ahsan et al., 2009a; Matter et al., 2009; Kagdi et al., 2012) or source code (Linares-Vásquez et al., 2012). In general, these approaches use machine learning techniques to automatically suggest a list of appropriate developers for a new incoming CR.

* Corresponding author. Tel.: +55 4896627336.
E-mail addresses: yguarata@gmail.com (Y.C. Cavalcanti), ivanmachado@dcc.ufba.br (I.d.C. Machado), pamsn@cin.ufpe.br (P.A.d.M.S. Neto), esa@cin.ufpe.br (E.S.d. Almeida).

http://dx.doi.org/10.1016/j.jss.2016.01.038
0164-1212/

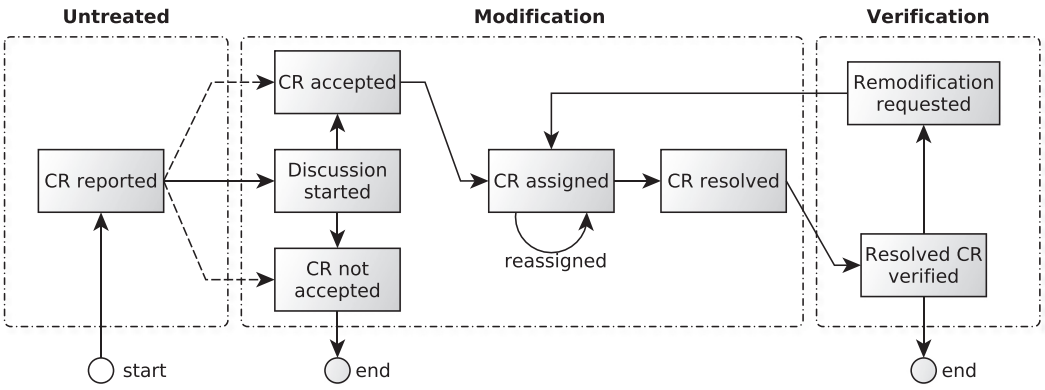

Fig. 1. CR workflow adapted from Ihara et al. (2009).

Despite the number of proposals, there is no empirical evidence about their applicability to real-world environments. To the best of our knowledge, most practitioners are still assigning CRs manually. Current approaches have not been adopted because of two main problems, as follows (Cavalcanti et al., 2014):

- They were designed to be autonomous, so that the software analysts do not have the control of the approach; this is, they cannot modify the behavior of the approach. Without such control, in turn, the approach cannot be properly calibrated. As a consequence, if its performance is not satisfactory, it is simply discarded.
- These approaches lack contextual information necessary to assign CRs properly. Software development companies might be highly dynamic, in terms of involved staff, e.g., developers move from project to project; developers can be hired/fired during project development; or they can even take a vacation or a day off. This dynamic influences the assignment of CRs. Thus, contextual information impacts the performance of automated approaches.

In this paper, we present a configurable approach developed to assign CRs which enables software analysts to control its behavior, as well as, it provides a mean to support contextual information necessary to perform effective assignments in dynamic environments. The approach relies on Rule-Based Expert System (RBES) and machine learning techniques.

The main ideas for this work come from our past three publications (Cavalcanti et al., 2013b, 2013c, 2014). In Cavalcanti et al. (2014), our approach was introduced but with less details. Thus we added more information about the approach, such as its architecture, implementation, and machine learning techniques. From Cavalcanti et al. (2013c), which is a survey with software developers, we selected the specifics results that helped us to propose the semi-automated solution. Then, we used results from the work (Cavalcanti et al., 2013b), which is an extensive mapping study on CR repositories issues, to elaborate the related work specific to the topic of assigning CRs.

Besides putting together these work, we also provided an extended experimental study of the proposed approach. According to the experiment performed, which compared our approach against other solution based solely on machine learning algorithm, we observed that ours improved the accuracy of assignments by 46.5%.

The remainder of this paper is organized as follows: Section 2 provides some background on CR management; in Section 3 we present the questionnaire-based survey; Section 4 presents the proposed approach to semi-automate the assignment of CRs; Section 5 describes the empirical validation performed to evaluate the proposed approach; Section 6 describes related work; and Section 7 concludes this work.

2. Change request management

A CR is a software artifact that describes a defect to be fixed, an adaptive or perfective change, or a new functionality to be implemented in a software system (Cavalcanti et al., 2013a). They are managed with the support of specific software systems which we simply refer as *CR repositories*. Examples of such repositories are Bugzilla (Bugzilla, 2013), Mantis (Mantis Bug Tracker, 2013), RedMine (Redmine, 2013), and Trac (The Trac Project, 2013). The CR repositories play a fundamental role in the maintenance process, being actually a focal point for communication and coordination in the software project (Bertram et al., 2010). In fact, the CR artifact is the primary unit of work in many software development projects (Anvik and Murphy, 2007).

The information carried in CRs is an important project documentation and history, since CRs hold healthy information about software evolution and maintenance activities. Indeed, during the life cycle of a given CR, it is common for discussions to take place concerning fixing alternatives and software design considerations (Bertram et al., 2010).

When a new CR is created, it is supposed to follow a specific workflow, which is implemented as a state machine in the CR repository. Fig. 1 shows a generic workflow for the purpose of explanation, adapted from Ihara et al. (2009). It is worth to mention that most CR repositories enable customizations, to meet project needs, although the one showed in Fig. 1 is very representative (Ihara et al., 2009). The workflow encompasses three phases, as shown in Fig. 1: *Untreated Phase*, *Modification Phase*, and *Verification Phase*.

Initially, in the *Untreated Phase*, the CR is created and reported in the project CR repository. More specifically, each CR stores different information fields which are essential to understand and implement the request. For instance, it stores a description of the change, the type of the change (e.g., defect or enhancement), the target component impacted by the change, the version of the software, among many others that can be defined in the CR repository.

In the *Modification Phase*, the CR can be either accepted or not, and the discussion on the CR acceptance and implementation takes place. A CR repository usually supports the discussion, by enabling submitters to use a comment field to enter messages regarding the CR under analysis. Notice that it is not necessary to have a discussion before deciding whether the CR will or will not be accepted. There are many reasons for not accepting a CR, such as: poor descriptions, redundancy, i.e., whenever the reported issue refers to an existing CR, or the reported CR is not planned to be fixed, etc.