Information and Software Technology 56 (2014) 1377–1389

Contents lists available at ScienceDirect

Information and Software Technology

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

Why software repositories are not used for defect-insertion circumstance analysis more often: A case study

Lutz Prechelt [a,*], Alexander Pepper [b]

[a] Freie Universität Berlin, Berlin, Germany
[b] Infopark AG, Berlin, Germany

ARTICLE INFO

Article history:
Received 7 October 2013
Received in revised form 1 May 2014
Accepted 1 May 2014
Available online 10 May 2014

Keywords:
Mining software repositories
Version archive
Bug tracker
Root cause analysis
Bug
Bugfix

ABSTRACT

Context: Root-cause analysis is a data-driven technique for developing software process improvements in mature software organizations. The search for individual process correlates of high defect densities, which we call *defect insertion circumstance analysis* (DICA), is potentially both effective and cost-efficient as one approach to be used when attempting a general defect root cause analysis. In DICA, data from existing repositories (version archive, bug tracker) is evaluated largely automatically in order to determine conditions (such as the people, roles, components, or time-periods involved) that correlate with higher-than-normal defect insertion frequencies. Nevertheless, no reports of industrial use of DICA have been published.
Objective: Determine the reasons why DICA is not used more often by practitioners.
Method: We use a single-case, typical-case, revelatory-type case study to evaluate in parallel the importance of six plausible reasons (R1–R6). The case is based on 11 years of repository data from a small but mature software company building a product in the high-end content management system domain and describes a four person-months effort to make use of these data.
Results: While DICA required non-negligible effort (R3) and some degree of inventiveness (R2), the most relevant roadblock was insufficient reliability of the results (R6) combined with the difficulty of assessing this reliability (R5). We identify three difficulties that led to this outcome.
Conclusion: Current repository mining methods are too immature for successful DICA. Gradual improvements are unlikely to help; different principles of operation will be required. Even with such different techniques, issues with input data quality may continue to make good results difficult-to-have.

1. Introduction

Mining software repositories (MSR) is a set of techniques that exploit the data stored in existing databases such as source code version archives or issue tracking databases in order to obtain relevant insights (general or specific) about software products or software development processes. One of the potentially useful application areas when mining software repository data is understanding defect insertion and defect removal processes. More specifically, a process that we will call *defect insertion circumstance analysis* (DICA) attempts to identify correlates of defect insertions: when, where (contexts), and how they happen and who makes them happen. Such insights can potentially be used to make valuable process improvements.

The Case Study in the sense of Yin [28] is a research method most suitable when "a 'how' or 'why' question is being asked about a contemporary set of events over which the investigator has little or no control".

1.1. Research question

The present article is a case study aimed at the research question formulated in the title: Why aren't MSR techniques used for DICA more often than they are? The question is based on the observation that, although the MSR community strives to address and include practitioners, there are not many reports on MSR usage from practitioners – which may be understandable for some types of MSR but appears surprising for DICA.

We answer it by investigating a single, arguably common (see below) case of industrial DICA as it occured in its real context.

1.2. Research contribution

The answer we found for the research question is twofold: First, there appears to be no affordable method for assessing the

* Corresponding author. Tel.: +49 30 838 75115.
E-mail addresses: prechelt@inf.fu-berlin.de (L. Prechelt), alexander.pepper@infopark.de (A. Pepper).

http://dx.doi.org/10.1016/j.infsof.2014.05.001
0950-5849/

reliability of the results obtained from a DICA. As a consequence, practitioners find DICA to be too risky to be worth its substantial effort. Second, the reliability of these results appears to be low. As a consequence, practitioners cannot expect sound answers from a DICA and will hence not find it sufficiently valuable.

Our article makes the following research contributions:

- It presents results of a 4-person-month DICA attempt for a large 11-year industrial software repository performed by practioners at software company Infopark. We are not aware of any other such report from an explicitly described industrial setting.
- It sorts out, at least roughly, the relative contributions of five different potential reasons (R2–R6) for not using DICA more often.
- It demonstrates the dominant weight of the two reasons mentioned above. The second of these is known in the MSR community and is being worked on, but the first, although rather fundamental, does not currently receive much attention at all.

1.3. Structure of this article

Section 2 introduces the domain of study. It introduces terminology in Section 2.1, explains the basics of MSR in Section 2.2, describes the procedure and potential benefits of DICA in Section 2.3, and discusses the research question's foundation: whether DICA is indeed used rarely (Section 2.4).

Section 3 explains the design of the case study. It introduces the Infopark DICA case (Sections 3.1 and 3.2), lists possible reasons why DICA is performed rarely (Section 3.3) which we will use as the propositions to be investigated by the case study, describes the sources of evidence and the forms of triangulation used during the case study (Section 3.4), and explains why a single-case study is satisfactory in this particular situation (Section 3.5).

Sections 4–7 describe the four major phases of Infopark's DICA attempt:

- establishing bugfix links,
- discriminating true defects from other "bugs",
- mapping defect corrections to defect insertions, and finally
- performing the actual DICA.

Each of these sections describes what Infopark did, what problems it encountered, how it handled those problems, what results it obtained, and then interprets these facts for the purposes of the case study. Section 8 discusses threats to validity.

Section 9 discusses in how far the results should be viewed as new when considering related work by other researchers and Section 10 presents conclusions.

2. The domain of study: MSR and DICA

2.1. Terminology

We choose our terms such as to make the discussion in the present article simpler; these definitions are not intended to be fit for general purposes. In particular, we constrain our discussion to phenomena observable on the level of program source code, because that is the sort of information our data sources provide.

Many of these terms talk about fuzzy phenomena so some of the definitions are unavoidably vague. This fuzzyness is in fact an important phenomenon in our case study, but resolving it is not our goal, so we do not aim at making the definitions maximally precise.

- A *change* is a set of additions, deletions, and modifications to existing software that are being checked into the source code version archive together. It is represented by the check-in transaction's source code delta and its commit message.
- A *defect* is a property of source code that triggers avoidable rework (possibly outside the observed timeframe).
- *Rework* is any modification performed on a section of program source code that was written and checked in earlier. In our analysis, the unit of rework is a single change (check-in transaction). We call rework *avoidable* if the need to make that change was in principle known at the time of the original work. We call rework *unavoidable* if that need arose only later or could (from the point of view of the software developers) be known only later.
- An *issue* is a property of software that is addressed in unavoidable rework. In practice, it is often difficult to discriminate defect and issue, which turns out to be important in our study.
- *Bug* is a synonym for either a defect or an issue in everyday software developer language and the discrimination is often not made. A *bugtracker entry* is an entry in a change request database and addresses either a defect or an issue.
- A *bugfix* is rework (specifically: a change) that is intended to partially or fully resolve the defect or issue described by a bugtracker entry (even if that intention is not achieved). A *bugfix link* is a pair of a bugtracker entry and its corresponding bugfix.
- A *defect correction* is a bugfix whose bugtracker entry describes a defect (rather than an issue).
- A *defect insertion* is a change that introduces one or more defects into the software. Note that a defect correction may introduce new defects as well.

2.2. Mining software repositories (MSR)

In mature software processes, it will often be useful to employ quantitative data obtained during process execution to optimize day-to-day project control and to spot general improvement opportunities for the process as a whole [12]. Such behavior is for instance suggested by the process areas of CMMI levels 4 and 5 [5], in particular the Causal Analysis and Resolution level-5 process area and its Determine Causes of Selected Outcomes goal. Unfortunately, obtaining and analyzing suitable data can be costly. One attractive approach for overcoming this cost problem would be using data that is collected anyway and automating its analysis [6]. This approach has been followed for a number of years by a community working on "mining software repositories" (MSR) that formed after some initial works in the 1990s [11,10] and holds a yearly workshop/conference called MSR since 2004.[1]

MSR is defined as "analyz[ing] the rich data available in software repositories to uncover interesting and actionable information about software systems and projects" (MSR 2014 call for papers) and the repositories in question are for instance "source control systems, archived communications between project personnel, and defect tracking systems" (MSR 2014 call for papers), requirements management databases, or project planning databases.

Topics in MSR include general infrastructure tasks such as data extraction and cleansing as well as many types of applications, for instance "characterization, classification, and prediction of software defects", "analysis of change patterns and trends", "prediction of future software qualities", building "models for social and development processes", "models of software project evolution", and reliability models, or supporting "search-driven software development" (all from MSR 2014 call for papers).

[1] http://www.msrconf.org, all MSR calls for papers can be found here. The references and quotations from this website are as of 2012-04-26.