The Journal of Systems and Software 115 (2016) 102–110

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Modeling and analysis of reliability of multi-release open source software incorporating both fault detection and correction processes

Jianfeng Yang[a], Yu Liu[b,*], Min Xie[b], Ming Zhao[c]

[a] Faculty of Information Engineering, Guizhou Institute of Technology, Guiyang, China
[b] Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China
[c] Faculty of Engineering and Sustainable Development, University of Gävle, Gävle, Sweden

ARTICLE INFO

Article history:
Received 6 September 2015
Revised 22 December 2015
Accepted 14 January 2016
Available online 6 February 2016

Keywords:
Software reliability
Multiple upgrading
Fault correction process

ABSTRACT

Large software systems require regular upgrading that tries to correct the reported faults in previous versions and add some functions to meet new requirements. It is thus necessary to investigate changes in reliability in the face of ongoing releases. However, the current modeling frameworks mostly rely on the idealized assumption that all faults will be removed instantaneously and perfectly. In this paper, the failure processes in testing multi-release software are investigated by taking into consideration the delays in fault repair time based on a proposed time delay model. The model is validated on real test datasets from the software that has been released three times with new features. A comprehensive analysis of optimal release times based on cost-efficiency is also provided, which could help project managers to determine the best time to release the software.

1. Introduction

The software industry is growing rapidly and has become very competitive. As a result, many software developers are cutting back their schedules to ensure prompt delivery and developing new features to keep their products competitive. This is especially true for large and complex software. After a release, reported faults in previous versions will be removed and new functions may be designed to meet new requirements in new versions. Developers generally pay greater attention to balancing competition in the market, and thus risk quality because of the short software lifecycle. The upgradation process constitutes a challenge for software companies looking to produce highly reliable software and ensure the release time is on schedule.

Over the past four decades, researchers have studied a variety of methods to assess software reliability. One of the most widely investigated and applied of those methods is the software reliability growth model (SRGM) (Lyu, 2007; Amin et al., 2013; Febrero et al., 2014; Yamada, 2014). Most SRGMs utilize the fault data collected during the test process to describe the stochastic behavior of the software fault detection process (FDP) with respect to time, and it is reasonable to assume that the fault counts in each time interval are mutually independent of each other (Amin et al., 2013). Non-homogeneous Poisson process (NHPP) model is considered as one of the most effective models (Goel and Okumoto, 1979; Lyu, 1996; Ohishi et al., 2009). They have been successfully applied in many software projects to manage tests and predict operational reliability (Jeske and Zhang, 2007; Lin and Huang, 2008; Rana et al., 2014). They have also been utilized in making critical decisions, such as those involved in cost-benefit analysis, resource allocation, and release-time determination (Peng et al., 2013; Park and Baik, 2015; Wang et al., 2015).

Furthermore, a number of specific SRGMs have been proposed for investigating the reliability of Open Source software (OSS), which is a growing area of software development and applications. For example, Tamura and Yamada (2013) propose a method of software reliability assessment for the embedded OSS with flexible hazard rate modeling. Pachauri et al. (2013) blended fuzzy set theory with software reliability measurement and total cost analysis, and Gratus and Pratibha (2013) proposed an approach for carrying out pre-statistical data analyses based on assessment of software's reliability metrics. Luan and Huang (2014) proposed an improved Pareto distribution model for analyzing the failure process, although their method is confined to ungrouped data.

In this paper, the failure process in testing multi-release software is further explored by taking into consideration a delay in the fault repair time based on the time-delay model proposed by Wu et al. (2007). Both fault-correction and detection processes are considered. It is assumed that the faults in a new version comprise both undetected faults in a previous version and new faults

* Corresponding author. Tel.: +852 56286295.
E-mail address: lyu.12@my.cityu.edu.hk (Y. Liu).

http://dx.doi.org/10.1016/j.jss.2016.01.025
0164-1212/© 2016 Elsevier Inc. All rights reserved.

introduced during the development process of the new version. A framework for assessing the expected number of remaining faults in each version is proposed and the optimal release time for each version is also investigated.

The remainder of this paper is organized as follows. Section 2 outlines the proposed framework for multi-release software modeling. In Section 3, the parameter estimation with Least Square Estimation (LSE) is developed and the optimal release strategy for such software is discussed. Section 5 demonstrates the application of the proposed models with a three-release dataset collected from a practical OSS test process and presents the results of optimal release time analysis. Finally, the conclusion is given in Section 6.

2. Literature review and further discussion of the multi-release problems

2.1. Modeling the multi-release situation

Most of the existing SRGMs focus on the software development process of only a single version. It is thus necessary to investigate changes in reliability arising from ongoing releases, which is a rather complex problem as usually there are many reasons for a new release. Several studies have been carried out in this regard in the literature. For example, Smidts et al. (1998) applied software failure data from a previous release to perform reliability estimation on a current release, and developed an early prediction model with a proposed Bayes framework using subjective and/or objective data from older projects. Hu et al. (2011) considered a scenario in which a software development team develops, tests, and releases software version by version, and proposed a number of practical assumptions. Li et al. (2011) later proposed a model that focuses on OSS and regards changes in testing effort with time as a hump-shaped curve. Recently, Pachauri et al. (2015) proposed a modeling framework considering the inflection S-shaped fault reduction factor and extended this model into multi-release software.

Many other factors, such as fault severities and test resources, are also incorporated into the modeling of multi-release software. Different severities describing the difficulty of correcting faults are considered during the upgrade process by Garmabaki et al. (2011), who assumed that the severity of the dormant faults in previous versions may change in subsequent versions. Kapur et al. (2012) discovered that some dormant faults in previously released versions can be removed in the tests of subsequent versions, and proposed a chain of SGRMs that take into account testing resources with a Cobb Douglas production function to optimize upgrade modeling and release time prediction.

2.2. Modeling fault correction delay

Most of the aforementioned modeling frameworks operate under the idealized assumptions that all faults are removed instantaneously and perfectly and that the expected number of removed faults is the same as the expected number of detected faults. In fact, time is always required for removal, and the expected number of removed faults at any given time is smaller than the expected number of detected faults (Gokhale et al., 2004). Accordingly, some researchers also take into account the fault correction process (FCP) and use corrected fault data to represent the correction time delay. Modeling both FDP and FCP requires more information from software testing records but improves estimation and prediction results. Schneidewind proposed an approach to FCP modeling that uses a constant delayed FDP (Schneidewind, 2001). He assumed that the rate of fault correction is proportional to the rate of failure detection.

However, because the FCP is heavily dependent on the FDP and there are many faults that have been detected but are still waiting for correction in some applications, the model usually underestimates the remaining faults in the code. Lo and Huang (2006) proposed an integrative method for analyzing the detection and correction processes using a differential equation. Wu et al. (2007) extended Schneidewind's model to a continuous version by substituting a time-dependent delay function for constant delay. Based on the aforementioned NHPP-based FDP and FCP modeling framework, both LSE and MLE (maximum likelihood estimation) approaches have been proposed. In addition, Hu et al. (2007) developed a neural networks configuration approach with an extra factor characterizing the dispersion of prediction repetitions used to simultaneously model the FDP and FCP. Huang and Hung (2010) later applied queuing models to describe the two processes with multiple change points. Incorporating a testing effort function and imperfect debugging, Peng et al. (2014) recently proposed a framework for analyzing both processes. Recently, Gaver and Jacobs (2014) proposed a queue model based on different failure mode assumptions.

3. Multi-release modeling framework for FDP and FCP

3.1. Single-release modeling framework for FDP and FCP

For single-version software, the method of modeling FDP is like the traditional NHPP SRGM in which the cumulative number of detected faults, $N(t)$, is assumed to follow a Poisson distribution with mean value function (MVF) $m_d(t)$, i.e.,

$$P\{N(t)=n\}=\frac{m_d^n(t)}{n!}e^{-m_d(t)}. \tag{1}$$

According to the basic assumption of fault removal, the MVF can be given by

$$\begin{cases}\dfrac{dm_d(t)}{dt}=\lambda_d(t)=\dfrac{F'(t)}{1-F(t)}[a-m_d(t)] \\ m_d(0)=0\end{cases}, \tag{2}$$

where $\lambda_d(t)$ refers to the failure rate during the test process and $F(t)$ is a cumulative distribution function. In solving the above differential equation, the MVF can be written as

$$m_d(t)=aF(t). \tag{3}$$

When $F(t)$ is assigned to an experiential distribution, it becomes the well-known GO model (Goel and Okumoto, 1979):

$$m_d(t)=a[1-\exp(-\gamma t)]. \tag{4}$$

The fault correcting process can be modeled as a stochastic time delay (obeys a random distribution of $G(t)$) of the FDP, and then delayed failure rate (and fault correcting rate) λ_c^* and delayed MVF m_c^* are as follows:

$$\lambda_c^*=\begin{cases}\lambda_d(t-\Delta t), & \Delta t\le t\\ 0, & \Delta t>t\end{cases} \tag{5}$$

$$m_c^*=\begin{cases}m_d(t-\Delta t), & \Delta t\le t\\ 0, & \Delta t>t\end{cases} \tag{6}$$

According to the approach proposed by Dai et al. (2007), $\lambda_c(t)$ can be the expectation of the delayed failure rate, that is,

$$\lambda_c(t)=E[\lambda_c^*]=\int_0^t\lambda_d(t-x)\cdot g(x)dx \tag{7}$$