Optimal Distribution of Software Testing Time Considering Multiple Releases

QINGPEI HU¹, RUI PENG²*, and GREGORY LEVITIN³,⁴
¹Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
²University of Science & Technology Beijing, Beijing, China
³The Israel Electric Corporation Ltd, Haifa, Israel
⁴Collaborative Autonomic Computing Laboratory, School of Computer Science, University of Electronic Science & Technology of China, Chengdu, China

(Received on June 13, 2012, revised on July 19, 2012)

Abstract: This paper considers a software development scenario where a software development team develops, tests and releases software version by version. A modeling framework is proposed to study the expected number of remaining faults in each version. The optimal development time and testing time for each version are also studied.

Keywords: Software reliability, software testing, multiple releases, single release, version management

1. Introduction

In order to track the growth of software reliability during the testing phase, numerous software reliability growth models (SRGMs) have been proposed during the past three decades [1-3]. Most SRGMs consider software development process during which only one software version is developed, tested and released.

In practice many software are developed and released version by version in order to better capture the market. For example, the first version of Matlab was released in 1984 as Matlab 1.0, the second version was released in 1986 as Matlab 2, ..., and the latest version was released in April 2011 as Matlab 7.12 (also named Matlab R2011a). Usually a new version is developed based on the previous version through correcting detected defects in the previous version and adding some new functions for new requirements. It is also presented in [4] that “A useful characteristic of software products is that not all features need to be released simultaneously. Instead, there is the possibility of offering additional features as upgrades to an existing product”. The software evolution process has also been studied in some other papers [5, 6].

In this work, we consider a scenario where a software development team develops, tests and releases software version by version. Section 2 presents the model. Section 3 studies the optimal development time and testing time for each version. Section 4 concludes.

2. The Method

We assume that totally \( N \) software versions are to be developed. Each version of the software is developed based on the previous version.

It is assumed that the development of the \( i \)-th version starts from \( t_{id} \) and ends at \( t_{id} \). After \( t_{id} \), the version undergoes a testing phase until time \( t_{ir} \) before it is released, during
which faults can be detected and corrected. Once version \(i\) is released at \(t_i\), the development of version \(i+1\) starts, i.e., \(t_{i+10} = t_i\). During the development phase of version \(i+1\), the field testing on the operational version \(i\) continues until \(t_{i+1d} = t_{ir}\). For each detected fault, it is reported and removed from both version \(i\) and version \(i+1\). The field testing of the final version lasts until all the faults are detected and corrected.

\[
\begin{align*}
  t_{i0} & = 0 \\
  t_d & = t_f \\
  t_{is} & = t_{id} \\
  0 & = t_{20} = t_{1f} \\
  t_{2d} & = t_{2r} \\
  t_{2s} & = t_{2d} \\
  \vdots
\end{align*}
\]

**Figure 1: Software Development Process with Multiple Releases**

Figure 1 illustrates the software development with multiple releases. We use \(a_i\) to denote the expected number of faults introduced during the development of version \(i\), that is, from \(t_{i0}\) to \(t_{id}\). The fault detection process during each version’s testing phase is described with non-homogeneous Poisson process model. We use \(b_i\) to denote the fault detection rate during testing phase before the release of any version \(i\) (from \(t_{id}\) to \(t_{ir}\)) and \(c_i\) to denote the fault detection rate during the field operations after the release of any version \(i\) (from \(t_{ir}\) to \(t_{is}\)).

The mean number of faults in version \(i\) (\(i > 1\)) from \(t_{id}\) to \(t_{is}\) can be obtained as

\[
N_i(t) = \left\{ \begin{array}{ll}
\sum_{j=1}^{k} a_j \exp[-b(t-t_{jd}) + \sum_{k=j+1}^{k-1} (t_{kr} - t_{jd})] \exp[-c(\sum_{j=1}^{k-1} (t_{ks} - t_{kr}))] & , t_{jd} \leq t \leq t_{ir} \\
\sum_{j=1}^{k} a_j \exp[-b(t-t_{jd})] \exp[-c(t-t_{ir}) + \sum_{k=j+1}^{k-1} (t_{ks} - t_{ir}))] & , t_{ir} < t \leq t_{is} \\
\end{array} \right.
\]

**3. Optimal Distribution of Testing Time**

This section assumes that the software development team has multiple alternatives for the development time and testing time for each version. In this case, the objective is to minimize the total cost function, which captures the fixed cost for software development, the opportunity cost for delaying the release of each version, the cost of correcting each fault during testing, and the cost of correcting each fault in the field.

The development time of any version \(k\) (\(k=1, \ldots, N\)) can be expressed as \(t_{kp} = t_{jd}\) and \(t_{kp+1} = t_{kd} - t_{ir}\) for \(k=2, \ldots, N\). The expected number of faults introduced during version \(i\) development \(a_i\) is assumed to be a decreasing function of \(t_{jd}\). In practice the software developers tend to make more faults when the schedule is tighter. The testing time on any version \(k\) (\(k=1, \ldots, N\)) can be expressed by \(t_{kt} = t_{kd} - t_{kr}\).

According to (1), we have

\[
\begin{align*}
N_i(t_{kd}) & = \sum_{j=1}^{k} a_j(t_{yp}) \exp[-b(t_{kd}) + \sum_{k=j+1}^{k-1} (t_{kr} - t_{jd})] \\
N_i(t_{kr}) & = \sum_{j=1}^{k} a_j(t_{yp}) \exp[-b(t_{kr}) + \sum_{k=j+1}^{k-1} (t_{ks} - t_{kr})] \\
N_i(t_{ks}) & = \sum_{j=1}^{k} a_j(t_{yp}) \exp[-b(t_{ks}) + \sum_{k=j+1}^{k-1} (t_{kp} - t_{yp})], \; k < N
\end{align*}
\]

The cost function is formulated as

\[
C(t_{ip}, \ldots, t_{kp}, t_{ip}, \ldots, t_{kp}) = c_i \sum_{j=1}^{N_i} (t_{ip} + t_{kp}) + \sum_{j=1}^{N_i} c_j [t_{ip} + \sum_{k=j+1}^{k-1} (t_{ip} + t_{kp})] + c_i \sum_{j=1}^{N_i} [N_i(t_{ip}) - N_i(t_{kp})] + c_j \left\{ \sum_{j=1}^{N_i} [N_i(t_{ip}) - N_i(t_{kp})] + N_i(t_{kp}) \right\}
\]
where \( c_0 \) is the fixed cost per time unit, \( c_k \) is the cost of delaying the release of version \( k \) for a time unit, \( c_c \) is the cost for correcting a fault during testing phase, and \( c_f \) is the cost for correcting a fault in the field.

The optimal distribution of testing time problem can be formulated as

\[
(t_{1p}^*, ..., t_{Np}^*, t_{1t}^*, ..., t_{Nt}^*) = \arg \min_c [C(t_{1p}^*, ..., t_{Np}^*, t_{1t}^*, ..., t_{Nt}^*)]
\]

(6)

In practice, each \( t_{ip} \) and \( t_{it} \) usually have some ranges. The optimization problem formulated by (6) can be solved using some constrained optimization toolbox, such as “optimtool” in Matlab.

4. Conclusions

This paper considers a scenario when a software development team develops, tests, and releases software version by version. Each version is developed based on the last version. A framework is proposed to model the remaining faults in each version of the software. The optimal development time and testing time for different versions which minimizes a composite cost function is also studied.

Acknowledgement: This work is Supported by National Natural Science Foundation of China, and Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

References


Qingpei Hu received his Ph.D. degree from Industrial & Systems Engineering Department, National University of Singapore in 2007. He is currently an assistant professor in Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China. His research is mainly on software reliability modeling and analysis.

Rui Peng received his Ph. D degree from Industrial & Systems Engineering Department, National University of Singapore in 2011. He is currently an assistant professor at Dongling School of Economics & Management, University of Science & Technology Beijing. His research interests include software reliability, networked system reliability, and defense strategies.

Gregory Levitin received his Ph.D. degree in Industrial Automation from Moscow Research Institute of Metalworking Machines in 1989. Dr. Levitin is presently a senior expert at the Reliability Department of the Israel Electric Corporation. His current interests are in operations research and artificial intelligence applications in reliability and power engineering.