计算机科学
COMPUTER SCIENCE

软件中总故障个数相关的不完美排错可靠性模型建模机理与述评

张策, 孙智超, 纪可行, 王金勇, 王宇彬

引用本文

张策, 孙智超, 纪可行, 王金勇, 王宇彬. 软件中总故障个数相关的不完美排错可靠性模型建模机理与述评[J]. 计算机科学, 2025, 52(6): 21-34.

ZHANG Ce, SUN Zhichao, JI Kexing, WANG Jinyong, WANG Yubin. Modeling Mechanism and Review of Imperfect Debugging Reliability Model Related to the Total Number of Faults in Software [J]. Computer Science, 2025, 52(6): 21-34.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240300061

软件中总故障个数相关的不完美排错可靠性模型建模机理与述评

张　策[1]　孙智超[2]　纪可行[1]　王金勇[3]　王宇彬[1]
1 哈尔滨工业大学(威海)计算机科学与技术学院　山东 威海 264209
2 华为南京研究所　南京 210000
3 山西大学自动化与软件学院　太原 030006

摘　要　挖掘可靠性研究中,软件故障总数对测试资源分配、可靠性变动影响以及最优发布等具有重要意义,但迄今为止鲜有从故障总数的角度进行可靠性研究。针对贴近真实测试环境的不完美排错等问题,对软件中故障总数相关的可靠性增长模型进行深入研究和系统述评。首先,对软件可靠性增长模型 SRGM(Software Reliability Growth Model)进行评述,给出研究主题、本质与技术内涵,引出软件中故障总数分析。从排错的不完全角度引入不同的新故障模型视角,建立不完美排错模型,分类研究多种情况下软件中故障总数与累积检测到的故障数量二者的变动情况。然后,从排错的不完全性与引入新故障的角度,建立统一的二元一阶不完美排错微分方程组描述软件测试过程,求解得到相应的故障总数与累积检测故障数量表达式。对上述两大类情况下不完美排错模型在多个真实计算机工程系统失效数据集上进行验证,从拟合与预测角度分析不同模型的性能,进而分析软件中故障总数对可靠性的影响。结果表明,故障总数对可靠性模型具有明显影响,其自身性能能够支撑可靠性的增长与性能提升。最后,指出了下一步研究挑战与亟待解决的问题。

关键词:可靠性;软件可靠性增长模型;可靠性建模;总故障个数;不完美排错
中图分类号　TP311

Modeling Mechanism and Review of Imperfect Debugging Reliability Model Related to the Total Number of Faults in Software

ZHANG Ce[1],SUN Zhichao[2],JI Kexing[1],WANG Jinyong[3] and WANG Yubin[1]
1 School of Computer Science and Technology,Harbin Institute ofTechnology,Weihai,Weihai,Shandong 264209,China
2 Huawei Nanjing Research Institute,Nanjing 210000,China
3 School of Automation and Software Engineering,Shanxi University,Taiyuan 030006,China

Abstract　In the reliability research,the total number of software faults is crucial for allocating testing resources,assessing reliability changes,and determining optimal release times. However,there has been limited research from the perspective of total faults counts thus far. This study delves deeply into reliability growth models related to the total number of faults in software,particularly in environments that closely mimic real testing scenarios,including imperfect debugging. Initially,the study reviews the software reliability growth model(SRGM),outlining its main themes,essence,and technical content,and introduces analysis of total failures in software. It incorporates models that introduce new faults from the perspective of imperfect debugging,and establishes an imperfect debugging model to categorize the dynamics of total failures and cumulative detected failures under various conditions. Subsequently,from the perspectives of imperfect debugging and the introduction of new faults,a unified binary first-order imperfect debugging system of differential equations is developed to describe the software testing process. Solutions are derived for expressions of total failures and cumulative detected failures. The performance of these models is validated against multiple real-world computer engineering system faults datasets,analyzing their fit and predictive capabilities,thereby assessing the impact of total failures on reliability variations. The results indicate that the total number of failures significantly influences the reliability models and supports the growth and performance enhancement of reliability. Finally,this paper highlights forthcoming research challenges and pressing issues that need to be addressed.

Keywords　Reliability,Software reliability growth model,Reliability modeling,Total number of faults,Imperfect debugging

到稿日期:2024-03-09　返修日期:2024-08-17
基金项目:国家自然科学基金(62373119);2021 年度山东省自然科学基金面上项目(ZR2021MF067);山西省基础研究计划(201801D121120);威海市科技发展计划项目(ITEAZMZ001807)
This work was supported by the National Natural Science Foundation of China(61473097),Shandong Province Natural Science Foundation(ZR2021MF067),Shanxi Province Basic Research Program(201801D121120) and Weihai Science and Technology Development Plan Project(ITEAZMZ001807).
通信作者:张策(zhangce@hitwh.edu.cn)

近年来，在新一代信息通信技术加速演进的带动下，软件产业发展迅猛，软件被赋予了新的使命，人类对软件功能作用寄予了更大的期望。相应地，软件的形态演化[1]得到了进一步丰富，单机版、网络版、移动版、大型网购软件平台、大型在线开放课程学习平台、大型社交平台、大型通信平台和开源软件等纷纷涌现，特别是云计算、大数据、物联网、人工智能、区块链、元宇宙等加剧了“软件定义一切”时代[2-4]的演进。软件已成为人们学习、生产、生活等方面的工具性帮手和抓手。大型复杂软件，工业软件，数值计算与仿真软件（特别是 MATLAB，CAD，CAE 等类型）、安全类软件、操作系统与数据库等基础软件日益成为亟需突破的“卡脖子”堵点。软件产业在国民经济和社会发展中的作用越来越重要，因此软件的质量问题，特别是可靠性属性等必须得到足够的重视。面对这些新变化、新趋势，软件自身的缺陷与质量问题[5-6]受到了更多的关注，其可靠、安全、稳定[7-8]成为更多研究者关注的焦点。可靠性是软件众多非质量属性中的重要构成，成为软件可信安全[9-11]的基础。软件作为人工制品，经过需求分析、概要设计、详细设计、编码、测试、部署、应用等阶段发布到市场或提供给用户。在这一过程中，由其内在设计带来的缺陷或者因编程开发而引入错误是不可避免的，使得可靠性问题始终是制约软件质量的重要因素。因此，关于软件可靠性的研究[12-16]，一直是计算机或软件工程等学科的重要理论内容，也是软件基础理论与工程研究中需要不断优化改进的问题。

以大型复杂软件为例，易知，软件发布前必须经过一定周期的测试与修改，只有达到发布基准后才能发布[17]。由于软件结构的复杂性、测试过程的随机性等多种不确定因素影响[18-22]，特别是不完美排错现象[23-25]的普遍存在，软件中故障总数具有较强的随机变动性。软件中的总故障个数（本文中表示为 $a(t)$）是一个未知变量，如何在测试阶段尽可能地降低总故障数量，继而提高软件可靠性，是增强软件自身质量领域的重要主题。深入研究软件中的总故障个数 $a(t)$，对于提高软件可靠性、评估软件测试成本，以及预测软件发布时间都具有重要的理论意义与现实价值。

在当前的软件可靠性研究中，单独研究软件中总故障个数的理论较为匮乏，直接的研究文献亦不多，这也使得从研究故障总数角度来寻求可靠性提升变得困难。软件可靠性可以借助数学模型来进行建模、度量与评测，其中软件可靠性增长模型 SRGM（Software Reliability Growth Model）[26-30]就是一类重要的工具。SRGM 的建模离不开软件中故障总数的参与，$a(t)$是必不可少的参数[31-34]，这已在众多的模型研究中得到了证实。软件中故障总数 $a(t)$是判断软件质量高低的一种指标，同时其数量的变化还可以描述排错中新故障引入情况[35]，可靠性质量、软件开发与测试资源的投入、软件发布时机等也与 $a(t)$紧密相关。

在可收集到的文献中，尤其在 SRGM 研究领域，虽然提出了较多的 $a(t)$模型，但研究重点大多集中在 SRGM 的建立与评测上，对 $a(t)$本身及其对模型的影响鲜有涉及。当前最大的困难在于，尽管提出的众多建模软件中有关于故障总数的数学模型，但由于真实的软件测试中并不知晓具体数值（现有发布的失效数据集[36]中并不包含此项），因此难以直接验证。本文的验证方法是在建立的软件故障检测或修复模型中，把故障总数作为累积检测到的故障数量 $m(t)$的一个参数，将 $m(t)$与真实的失效数据集中累积检测到的故障数量进行比较（拟合与预测），从而间接地表明故障总数的准确性。此外，本文中术语“故障”和“失效”不区分对待，都是指影响软件正常功能和性能的任何异常行为或性能下降。这种用法有助于简化讨论并集中于研究主要问题。

在前期大量研究的基础上，本文主要关注 $a(t)$对 SRGM 模型的影响，重点剖析 $a(t)$的建模与评估，通过建立统一的不完美排错模型观测不同 $a(t)$对可靠性带来的变化，进而衡量各种 $a(t)$的性能差异。同时，研究具备描述新故障引入能力的 $a(t)$的不同形式对 SRGM 模型的扰动影响。

本文的贡献着重体现在以下 3 个方面。

1）对软件中的故障总数进行了全面论述，系统地区分了其不同的类型，对其基本功能、函数模型、作用成效进行了深刻阐述，这是软件可靠性研究领域首次对此进行研究；

2）首次从软件中总故障个数的角度，建立统一的不完美排错框架模型，形成了覆盖总故障个数变化率多种形式紧密相关的软件可靠性增长模型；

3）通过在多个实际应用场景下的真实失效数据集上的综合实验，进行基于拟合度量与预测分析的影响以评测软件中总故障个数建模与验证，系统分析了软件中总故障个数对可靠性的模型的扰动影响，为深入剖析不同模型之间的差异和选择模型提供了有重要价值的参考借鉴。

本文第 1 章对 SRGM 研究内容、建模过程、研究主题与技术要点进行了精要论述，为软件中故障总数研究做好铺垫；第 2 章对软件中故障总数进行整体概述，给出了 5 种函数分类，通过实验初步得到 3 类曲线增长形状，为后续深入研究其对可靠性的影响提供基础；第 3 章重点从软件中故障总数发生变动的不同情况，建立了两类统一的不完美排错模型，并进一步衍生出具体故障总数函数支持的不完美排错特定模型；第 4 章在 12 个真实发布的失效数据集上进行综合实验，通过不同可靠性模型间、不同故障总数模型间的性能对比，分析故障总数对可靠性的影响；第 5 章指出了研究挑战与亟待解决的问题；最后总结全文。

1 SRGM 研究内容、技术分析

自 1978 年以来，非齐次泊松过程 NHPP（Non-Homogeneous Poisson Process）类型的 SRGM 一直是研究的主流，主要包括以 G-O 模型[37]及其改进为代表的指数型和以 Yamada 模型为代表的 S 型模型[38]，它们均是对故障检测与修复过程进行建模，进而求得能够表征软件可靠性的累积检测的故障数量（即 $m(t)$函数）。从检测与修复的故障数量角度来研究可靠性的增长是 SRGM 研究的切入点，显然，被检测到的故障不断被排除，可靠性势必会得到提高。

科研人员通过对测试过程的建模，得到$[0,t]$内累积检测的故障数量（这里标记为 $m(t)$），并利用真实的失效数据集进行验证。建模中，通常自行设定软件中总故障个数 $a(t)$的表达式，以及故障检测率[39-41]（这里标记为 $b(t)$）等。需要指出，假设差异是不同 SRGM 存在区别的根本原因，这种差异

导致最终求解得到的 $m(t)$ 存在巨大差异。应用 SRGM 进行可靠性分析是对可靠性进行建模和评测的主要手段。基于前期的大量研究[42-45]，图 1 为科研人员基于对测试过程的认知下进行 SRGM 建模的基本流程，涉及软件可靠性研究相关的测试过程、研究内容、SRGM 的效用以及相应的技术对应关系。图 1 对测试过程的认知、建立数学模型、求解与获得各重要函数表达式、确定软件成本以及软件最优发布为主要流程进行了阐释。

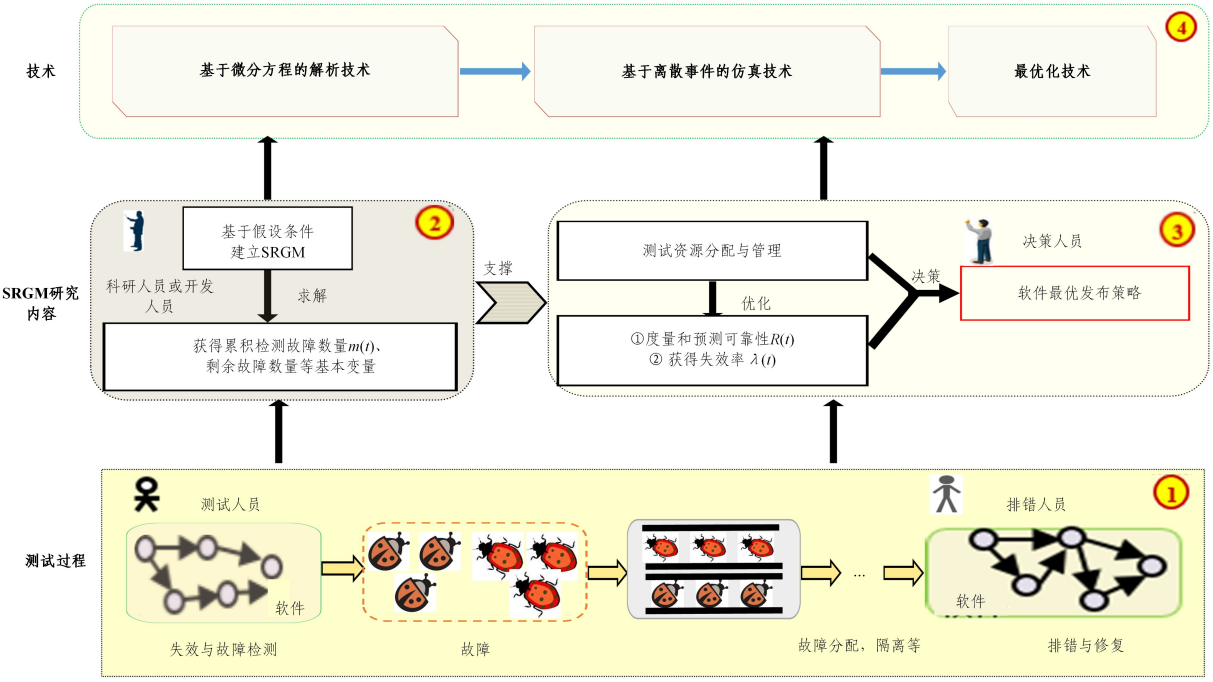


图 1　测试过程抽象、SRGM 建模与研究内容以及相应的技术

Fig. 1　Test process abstraction, SRGM modeling and research contents and corresponding technologies

目前，与 SRGM 紧密相关的研究领域整体上呈现出如下技术演进路线：传统基于微分方程（组）建模测试过程中故障检测至修复过程的解析方法（包含基于队列的研究）、基于仿真的分析方法（典型的如离散事件仿真）和最优化技术。

SRGM 的研究基本呈现出如下的发展态势，大致包括 6 个方面的研究主题，所采取的具体技术也有所不同，如表 1 所列。总体而言，SRGM 研究主题可以概括为 4 个层面，即 SRGM 本身关于模型建立用于评测可靠性的研究、基于测试资源分配进行成本管控以决定最优发布的研究、采用仿真技术对测试过程进行模拟的研究，以及对众多模型的决策选择研究。

表 1　SRGM 研究主题、本质与技术概要分析

Table 1　Summary analysis of SRGM research theme, essence and technology

SRGM 研究主题	研究主题概要分析	本质与技术
测试过程建模	通常基于微分方程或微分方程组的方法，建模描述[0，t]内故障检测或修复情况	建立故障检测、修复、引入等与测试时间的定量函数关系，即建立 SRGM——基于微分方程（组）技术
测试资源分配	通常基于微分方程求解得到或人为设定测试资源消耗的函数形式	在受限的资源约束下，建模测试工作量与花销，建立满足特定目标时的测试资源分配方案——最优化技术
测试成本管控	基于故障被检测与修复的情况，以及测试资源消耗的情况，建立成本表达式	对软件从测试至发布的成本支出建立数学模型，进行定量化研究——最优化技术
软件最优发布	通常基于对测试预期（例如，可靠性达到何种程度或测试成本不超过预期）的最优化建模，确定软件发布的最优时间	为达到多种预期目标的软件发布，建立受约束时的最优化模型，决定软件最优发布时机——最优化技术
测试过程仿真	区别于采用微分方程方法，对软件测试过程中的故障检测、修复和引入等环节进行模拟仿真，避免复杂的数学运算	对软件测试随机过程进行仿真——离散时间仿真技术
模型决策选择	通常基于对 SRGMs 的性能评价进行最优化抉择，给出最优的模型或排序	在众多模型范围内，基于模型的拟合与预测等指标建立多属性的优化模型，进行决策——最优化技术或多属性决策技术

软件可靠性研究涉及多种实际因素，具有复杂的随机过程特点，涵盖了完整的测试过程，包括至少如表 1 所列的研究内容和技术，是理论研究与可靠性工程和软件工程相结合的重要实践。

在 SRGM 的建模中，考虑到对测试过程认知与假设的不同而先后衍生出众多的模型，这些模型中均含有故障总数 $a(t)$ 这一关键性参变量，这已经成为可靠性研究领域的基本共识。例如，在早期的 G-O 模型中，建立了如式(1)所示的软件可靠性模型：

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t}=b\times(a-m(t)) \tag{1}$$

该模型认为[0，t]时间内累积检测的故障数量是 $m(t)$，并假定软件开始测试时内部的总故障数量是 a 常量，假设 t 时刻检测的故障数量（实际上是 t 时刻故障变化率）与当前软件中剩余的故障数量（即 $a-m(t)$）成正比（比例系数是 b），这样就可以得出式(1)，在 $m(0)=0$ 的初始条件下，进而求解得到 $m(t)=a(1-\mathrm{e}^{-bt})$。此外，软件在测试阶段的可靠性 $R(t|x)$ 可表示为：

$$R(t|x)=e^{-(m(x+t)-m(x))} \tag{2}$$

其中，x 为软件上次失效的时刻，为简化，设 x 从 0 开始计时，并将式(1)求解的结果带入式(2)，则可以得到(0，t]内软件的可靠性：

$$R(t)=e^{-a(1-e^{-bt})} \tag{3}$$

由此可见，故障总数不仅是可靠性建模中必不可少的要素，也决定着可靠性的增长，是对可靠性进行建模与度量的核心支撑。

特别指出，软件可靠性研究遵从如下默认规则：

1)SRGM 研究隐含基础是面向大规模软件系统；

2)前期所搜集的大量失效数据集 FDS(Failure Data Set)中，失效数据全部是知名公司或机构在开发大型软件测试过程中发布的。

2 软件中故障总数研究回顾

测试阶段，随着软件规模和复杂度的不断上升，软件中故障总数成为测试人员关注的一个重要方面。在当前可查证的 SRGM 研究文献中，软件中故障总数和故障检测率 FDR(通常用 $b(t)$来表示)均作为参与模型建立的必不可少的参数，是支撑可靠性研究的重要元素。在 SRGM 的研究中，软件中故障总数 $a(t)$是必不可少的参数，且故障总数 $a(t)$的变化可以描述排错中新故障引入的情况。

1)若 $a(t)=a$ 常量，则表明排错中无新故障引入。这种情况虽与真实的测试过程不相符，但可以简化问题研究的复杂程度，对于分析软件可靠性的增长趋势具有指导意义。

2)若 $a(t)=f(t)$，则是随着测试时间 t 的增函数。这种情况考虑了程序员在软件修复中存在引入新故障的可能，使得所建立的测试模型更加准确，但有时复杂的 $f(t)$会使得模型求解变得困难。

经过大量文献的研究，可以得到关于 $a(t)$的事实包括如下几个方面：

1)认为 $a(t)=a$ 是常量；

2)设定 $a(t)$为某种随着测试时间 t 或 $m(t)$或 $W(t)$递增的函数；

3)$a(t)>m(t)$，$[a(t)-m(t)]$表示软件中剩余的故障总数。

易知，在实际的测试过程中，由于存在新故障的引入，软件中总故障个数应该随测试与修改的进行发生增加性变化。因此，设定 $a(t)=a$ 并不合理，但在很多时候，可以简化问题研究的复杂度，且总体上可以得到明确的趋势性结论。

相比于 $m(t)$可以借助于失效数据集中公开的、真实的被检测故障数量(可以表示为 $n(t)$)来进行验证，$a(t)$难以直接验证，这成为了研究 $a(t)$的最大障碍。

软件测试与排错人员希望将全部故障排除掉，从而获得高可靠性，但这并非易事，也不现实。

1)软件中总的故障个数是未知的(虽然可以假定为有限或无限个数)，根本无法确定；

2)软件在排错过程中因排错的不彻底性以及引入新故障现象使得总故障个数增加。

为此，设定 $a(t)$为 t 的某种增函数形式较为常见，在前期研究[42-43]的基础上，这里将其归为五大类，如表 2 所列。

表 2 软件中总故障个数 $a(t)$分析

Table 2 Analysis of total number of faults $a(t)$ in software

类型	文献	$a(t)$	概要分析
常量类型	Huang et al.[46]，Yamada et al.[47]	$a(t)=a$	常量，无法描述修复中引入新故障的事实
乐观类型(有限故障)	Pham et al.[48]，Pham[49]	$a(t)=c+a(1-e^{-at})$	总故障个数有限：当 $t\to\infty$ 时，$\lim_{t\to\infty}a(t)=c+a$
悲观类型(无限故障)	Yamada et al.[50]，Pham[49]	$a(t)=ae^{at}$	总故障个数无限：当 $t\to\infty$ 时，$\lim_{t\to\infty}a(t)\to\infty$
	Yamada et al.[50]，Pham et al.[20,51]	$a(t)=a(1+at)$	总故障个数无限：当 $t\to\infty$ 时，$\lim_{t\to\infty}a(t)\to\infty$
	P-Z-2 模型[52]	$a(t)=a(1+rt)^2$	总故障个数无限：当 $t\to\infty$ 时，$\lim_{t\to\infty}a(t)\to\infty$
折中类型	Ahmad et al.[53]，Huang et al.[54]	$a(t)=a+am(t)$	由于 $m(t)$通常为增函数，但存在有限增长的可能，因此 $a(t)$增长也可能存有限度
	Peng et al.[55]	$a(t)=ae^{aW^*(t)}$	由于 $W^*(t)$通常为增函数，但存在有限增长的可能，因此 $a(t)$增长也可能存有限度；建模中融入测试工作量函数 TEF(Testing Effort Function)，使得测试资源的消耗能够被考虑，使得模型更加细腻
微分类型	Zhang et al.[56]	$\frac{da(t)}{dt}=k\frac{dm(t)}{dt}$	认为 $a(t)$的变化率与 $m(t)$的变化率成比例

由表 2 可知，$a(t)$与测试时间或检测到的故障数 $m(t)$成某种比例关系：

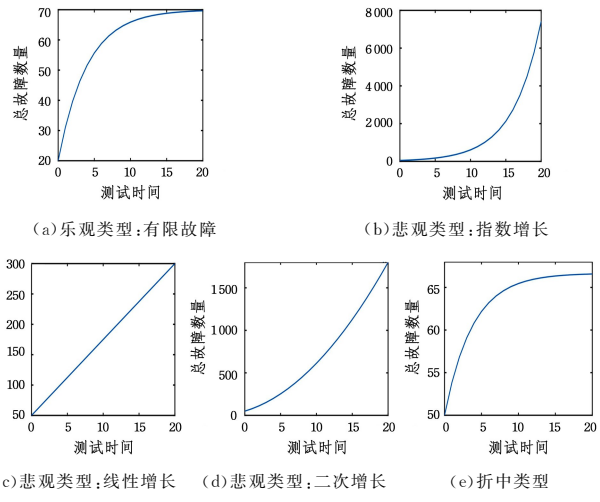
1)虽然有时设定 $a(t)=a$ 使得建模求得的 $a(t)$具有较好的性能，但真实的故障修复过程中会产生新故障引入软件中，因此设定 $a(t)$为常量是不合理的。

2)有研究设定 $a(t)$为 t 的表达式，本文认为这并没有本质地反映出 $a(t)$的变化情况，没有厘清 $a(t)$具体与何变量关系密切的问题。

3)显而易见，$m(t)$只是当前 t 时刻被检测出来的故障，是 $a(t)$的一部分而已，二者理应有较为密切的关联，因此有研究设定 $a(t)=a+\alpha m(t)$或$\frac{da(t)}{dt}=k\frac{dm(t)}{dt}$，有其一定的合理性。

4)另外，考虑到新故障引入是在软件修复过程中发生的，因此 $a(t)$与当前累积修复的故障数量 $r(t)$有较为密切的关联。易知，$r(t)\leqslant m(t)\leqslant a(t)$。因此，从 $r(t)$的角度来建模 $a(t)$亦具有一定的合理性。

图 2 给出了表 3 中 3 类 $a(t)$的基本形状，其中设定 $a=50$，$\alpha=0.25$，$b=0.35$，$r=0.25$，$t\in[0,20]$，$m(t)$采用经典的 G-O 模型(即 $m(t)=a[1-e^{-bt}]$)，仅用于展示曲线形状。



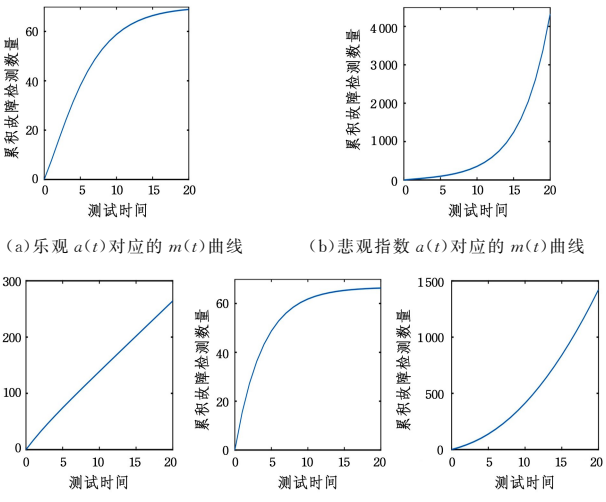
(a)乐观类型：有限故障　(b)悲观类型：指数增长

(c)悲观类型：线性增长　(d)悲观类型：二次增长　(e)折中类型

图 2　3 种类型软件故障总数 $a(t)$示意图：有限故障、无限故障、折中

Fig. 2　Schematic diagram of three $a(t)$ types: limited failure, wireless failure, compromise types

图 3 给出了表 3 中 3 类 $a(t)$所对应的 $m(t)$的基本形状，其中设定 $a=50$，$\alpha=0.25$，$b=0.35$，$c=20$，$p=1$，$r=0.25$，$t\in[0,20]$，仅用于展示曲线形状。



(a)乐观 $a(t)$对应的 $m(t)$曲线　(b)悲观指数 $a(t)$对应的 $m(t)$曲线

(c)悲观线性 $a(t)$对应的 $m(t)$曲线　(d)悲观二次 $a(t)$对应的 $m(t)$曲线　(e)折中 $a(t)$对应的 $m(t)$曲线

图 3　5 种 $a(t)$对应的累积故障检测数量函数 $m(t)$：有限故障、无限故障、折中类型

Fig. 3　$m(t)$ corresponding to the five $a(t)$: limited fault, wireless fault, compromise types

根据图 2 和图 3 可以看出，$a(t)$的曲线增长形状可大致分为 3 类：凸型指数增长、凹型指数增长与线性增长。同时，在故障检测率 $b(t)$与故障修复概率 $p(t)$固定为常数的情况下，$a(t)$与 $m(t)$在曲线形式上较为相似，即图像的凹凸性、增减性大多一致。

在有效的测试环境下，$m(t)$不断增长并与软件中故障综述 $a(t)$会更加接近($m(t)\leqslant a(t)$)，因此二者在曲线变动上存在着较强的相似性。图 3 中多数 SRGM 模型呈现指数型增长，这与文献[54]指出的当前绝大多数 SRGM 属于指数型增长趋势相符合。

3　软件中总故障个数相关的不完美排错模型建模与性能影响评测

3.1　软件中总故障个数相关的不完美排错模型建模

相比于完美排错模型[37,57]，真实的软件测试过程是受多种随机因素影响的复杂过程，典型地，排错的不完全性和新故障引入是两种真实存在的现象，是真实的不完美排错过程[58-65]的集中反映。排错的不完全性是一种不完美排错[66]，排错中引入新故障也是一种不完美排错[20]，这里认为不完美排错主要集中在排错的不完全或新故障引入上。因此，本节将从不完美排错的角度，建立软件中故障总数相关的 SRGM，研究故障总数对 SRGM 的影响。

首先，基于 SRGM 建立公共假设[37,67-72]，增加关于不完美排错相关的内容，形成如下假设：

1)软件失效满足 NHPP 过程；

2)在$(t+\Delta t)$内检测到的故障数量与当前软件中剩余的故障数量成比例；

3)软件修复过程中存在排错的不完全性和新故障引入现象。

3.1.1　直接设定总故障个数表达式

第一种情况下，研究人员直接设定故障总数为某函数形式，用于建模测试中故障总数的增长情况，为此增加以下假设：

1)软件排错的过程中，存在新故障被引入现象，软件中故障总数是测试时间 t 的某种增函数。

在假设 1)－假设 4)的基础上，可以得到如式(4)所示的方程组：

$$\begin{cases}\dfrac{\mathrm{d}m(t)}{\mathrm{d}t}=b(t)\cdot[a(t)-p(t)\cdot m(t)]\\ a(t)=f(t)\end{cases}\tag{4}$$

其中，第一个微分方程建模故障检测与不完全修复的过程模型，$b(t)$为 t 时刻的故障检测率，是当前时刻软件测试环境下对故障检测的整体描述；$p(t)$是故障修复比例，为 t 时刻的故障修复概率，可被设定为时间 t 的函数，表明测试中存在排错的不完全现象；$p(t)\cdot m(t)$表示 t 时刻累积修复的故障数量；第二个方程中 $a(t)$为 t 时刻软件总的故障个数，是随着测试时间 t 变化的动态函数。

式(4)中，当只考虑第一个子式时，在 $m(0)=0$ 的初始条件下，可求得 $m(t)$的表达式如下：

$$m(t)=\mathrm{e}^{-\int_0^t b(x)\cdot p(x)\mathrm{d}x}\int_0^t \mathrm{e}^{\int_0^y b(x)\cdot p(x)\mathrm{d}x}a(y)b(y)\mathrm{d}y\tag{5}$$

则当前的失效率 $\lambda(t)$为：

$$\lambda(t)=\mathrm{e}^{-\int_0^t b(x)\cdot p(x)\mathrm{d}x}\left[b(t)\cdot\left(p(t)\cdot\int_0^t \mathrm{e}^{\int_0^y b(x)\cdot p(x)\mathrm{d}x}a(y)b(y)\mathrm{d}y+a(t)\cdot\mathrm{e}^{-\int_0^t b(x)\cdot p(x)\mathrm{d}x}\right)\right]\tag{6}$$

测试阶段的软件可靠性 $R(t|x)$，即假定软件上一次的失效时间是 $x(x\geqslant 0,t>0)$，则在$(x,x+t)$内的软件可靠性可表示为：

$$R(t|x)=\mathrm{e}^{-[m(x+t)-m(x)]}\tag{7}$$

假定从 $x=0$ 开始，将式(5)带入式(6)，则可以得到可靠性 $R(t)$与 $a(t)$的关系，如下式所示：

$$R(t)=\mathrm{e}^{-a(t)[(1-\mathrm{e}^{-p(t)b(t)t})]}\tag{8}$$

可见，$a(t)$与可靠性$R(t)$紧密相关，$R(t)$是$a(t)$的函数。随着$a(t)$的增长变动，$R(t)$也随之增高。因此，研究$a(t)$可以成为掌握$R(t)$变动以及最优发布的重要方法。

$a(t)$主要被设定为如下5种形式，在$m(0)=0$和$a(0)=a$的初始条件下，可以求得$a(t)$的表达式如以下5种情况所示：

1)若$a(t)=c+a(1-e^{-\alpha t})$，在$m(0)=0$的初始条件下，则可求得$m(t)$如下：

$$m(t)=e^{-\int_0^t b(u)p(u)du}\int_0^t b(v)[c+a(1-e^{-\alpha v})]e^{-\int_0^v b(s)p(s)ds}dv \tag{9}$$

此时，$a(t)$为有限个数增长，$a(t\to\infty)=c+a$。

2)若$a(t)=ae^{\alpha t}$，在$m(0)=0$的初始条件下，则可以求得$m(t)$如下：

$$m(t)=e^{-\int_0^v b(s)p(s)ds}\int_0^t b(v)ae^{\alpha v}e^{-\int_0^v b(s)p(s)ds}dv \tag{10}$$

此时，$a(t)$为无限个数增长，$a(t\to\infty)\to\infty$。

3)若$a(t)=a(1+\alpha t)$，在$m(0)=0$的初始条件下，则可以求得$m(t)$如下：

$$m(t)=e^{-\int_0^v b(s)p(s)ds}\int_0^t b(v)a(1+\alpha v)e^{-\int_0^v b(s)p(s)ds}dv \tag{11}$$

此时，$a(t)$为无限个数增长，$a(t\to\infty)\to\infty$。

4)若$a(t)=a(1+rt)^2$，在$m(0)=0$的初始条件下，则可以求得$m(t)$如下：

$$m(t)=e^{-\int_0^v b(s)p(s)ds}\int_0^t b(v)a(1+rv)^2e^{-\int_0^v b(s)p(s)ds}dv \tag{12}$$

此时，$a(t)$为无限个数增长，$a(t\to\infty)\to\infty$。

5)若$a(t)=a+\alpha m(t)$，在$m(0)=0$的初始条件下，则可以求得$m(t)$如下：

$$m(t)=e^{-\int_0^t b(u)[p(u)-\alpha]du}\int_0^t ab(v)e^{-\int_0^v b(s)[p(s)-\alpha]ds}dv \tag{13}$$

此时，$a(t)$为无限个数增长，$a(t\to\infty)\to\infty$。

3.1.2 从微分方程角度设定总故障个数的变化率

第二种情况下，认为软件中故障总数的变动与累积检测的故障或修复的故障个数有关联[56]，为此在第一种情况下增加以下假设：

1)在$(t+\Delta t)$内引入的故障个数与当前检测或修复的故障个数成比例关系。

另一方面，新故障引入是在故障修复的过程中由于程序员对程序结构的改动而发生的，相较于$a(t)$变化率与检测的故障个数变化率成比例，$a(t)$变化率与修复的故障个数变化率成比例更加靠近真实的测试过程。因此，提出了考虑排错的不完全性与引入新故障的不完美排错模型：

$$\begin{cases}\dfrac{dm(t)}{dt}=b(t)\cdot[a(t)-p(t)\cdot m(t)]\\[2ex]\dfrac{da(t)}{dt}=\beta(t)\cdot\dfrac{d(p(t)\cdot m(t))}{dt}\end{cases} \tag{14}$$

其中，$m(t)$表示$[0,t]$内累计检测到的故障数量；$a(t)$表示软件中的总故障个数，由于考虑了引入新故障的情况，因此$a(t)$呈增长趋势。该微分方程建立的假设基础为：$[0,t]$内累计检测到的故障数量与当前剩余的故障数量成比例，比例为故障检测率$b(t)$。第1个子式建模刻画了故障检测与修复的模型，$p(t)$表示故障被成功排除的概率。第2个子式建模描述了故障修复中引入新故障的模型，分别描述了$a(t)$变化率与检测的故障个数变化率成比例，其中$\beta(t)$表示排错过程中引入的故障概率。

显然，当认为故障修复是完全时，即$p(t)=1$时，式(9)演变为完全修复且引入新故障的模型；当$0<p(t)<1$时，式(14)演变为不完全修复且引入新故障的模型；当$p(t)=1$且$\beta(t)=0$，或不考虑第2个子式时，式(14)演变为经典的G-O模型。

由于上式的求解较为复杂，为降低复杂度，考虑到求解的方便可行性和不影响模型的整体功效，不妨令$p(t)=p$。上式的边界条件是$m(t)=0$和$a(t)=a$。

令

$$x(t)=a(t)-p\cdot m(t) \tag{15}$$

由初始条件可得$x(0)=a(0)-p\cdot m(0)=a$。则由一阶齐次线性微分方程的通解可得$x(t)$为：

$$x(t)=ae^{-\int_0^t p\cdot(1-\beta(\tau))\cdot b(\tau)d\tau} \tag{16}$$

将式(15)代入式(14)的第1个子式可得：

$$\frac{dm(t)}{dt}=b(t)\cdot x(t) \tag{17}$$

将式(16)代入式(17)中可得：

$$m(t)=\int_0^t b(u)\cdot x(u)du=a\int_0^t b(u)\cdot e^{-\int_0^u p\cdot(1-\beta(\tau))\cdot b(\tau)d\tau}du \tag{18}$$

将式(18)代入式(14)的第2个子式可得：

$$a(t)=a(1+p\cdot\int_0^t\beta(u)b(u)\cdot e^{-\int_0^u p\cdot(1-\beta(\tau))\cdot b(\tau)d\tau}du) \tag{19}$$

这种情况下，关键的故障数量表达式$m(t)$和$a(t)$均是求解出来，且具体由3个参变量函数来决定，即故障检测率$b(t)$、故障修复概率$p(t)$，以及故障引入率$\beta(t)$。这些参变量函数可根据实际情况进行设定，为模型的框架性、统一性和柔韧性提供了支撑。这种情况下，SRGM中的$a(t)$由描述实际测试情况的多个因素变量决定，更加直接地反映了实际因素的变化情况，提升了$a(t)$建模的准确性。

3.1.3 讨论

1)关于$a(t)$的两种情况

第一种情况，$a(t)$被研究人员直接设定为某种函数表达式，它或随着测试时间t增长，或指数增长，或随着$m(t)$增长。这些类型从不同角度建模了$a(t)$的增长，整体上符合人们对测试过程的认识，具有一定的有效性。

相比于直接设定$a(t)$的表达式，第二种情况给出了$a(t)$的增长率与软件测试中累积检测和修复的故障数量的紧密关系，更加客观地描述了测试中故障检测与排除的内在本质。$da(t)/dt$与$dm(t)/dt$成比例，表明随着故障不断被检测出来的同时，新故障被不断引入；$da(t)/dt$与$d[p(t)\cdot m(t)]/dt$成比例，表明故障修复的过程中，新故障被不断引入。易知，新故障引入是在故障修复的过程中发生的，因此$da(t)/dt$与$d[p(t)\cdot m(t)]/dt$成比例更加符合真实的测试过程。

两种情况中，直接设定$a(t)=a+\alpha m(t)$，与设定$\dfrac{da(t)}{dt}=\beta(t)\cdot\dfrac{d(p(t)m(t))}{dt}$具有等价性，但由于参数不同，因此在具体的模型性能评价(拟合和预测)中会略有差异。

2)关于验证方式的有效性

关于所提出的模型验证，由于当前失效数据集中尚未发

布关于软件总故障的信息，因此无法直接验证，只能将建模求解得到的 $m(t)$ 与失效数据集中累积检测到的故障个数进行验证。显然，这种验证具有间接性。

自 1978 年 G-O 模型提出至今，有数百个 SRGMs 被提出，在其假设中均包含假设 1)，并据此建立了形式各异的微分方程。本文所建立的微分方程涵盖了当前研究的各种形式，是一种框架模型：

$$\frac{\mathrm{d}m(t)}{\mathrm{d}t}=b(t)\cdot[a(t)-p(t)\cdot m(t)] \tag{20}$$

该微分方程对测试过程中的故障检测（与修复）过程进行了建模，其中 $b(t)$ 是故障检测率函数，数值在 $(0,1)$；$p(t)$ 是故障修复概率，数值在 $(0,1)$；$a(t)$ 可取为常量，也可设定为 t 的某种增函数，或通过其增长率 $\mathrm{d}a(t)/\mathrm{d}t$ 的形式来进行设定。可以看出，根据需要可对 $b(t)$，$p(t)$ 和 $a(t)$ 进行设定，得到多种现存的模型。

上面关于 $a(t)$ 两种情况的 6 种形式，都是当前研究中对测试过程中关于 $a(t)$ 的某种认知，因此在一定程度上刻画了测试过程中故障总个数的变化情况。这样，在上面的框架式模型中，结合融入各种 $a(t)$ 模型所得到的 $m(t)$ 和 $a(t)$ 具有典型性和实用性，可以作为验证 $a(t)$ 有效性与否的一个视角。因此，这里所建立的模型具有典型性和合理性。

3）关于模型假设的必要性

在软件可靠性增长模型（SRGM）的研究领域中，研究者通过构建抽象模型来模拟软件测试和故障排除过程。这些模型的建立基于一系列精心设计的假设，旨在简化问题、促进模型构建和求解过程。值得注意的是，这些假设对于理论层面上对软件可靠性增长趋势的假设并无负面影响，反而为深入探究软件可靠性增长的核心机制提供了数学工具和方法论的支持。国内外的学术研究普遍采纳了这种研究方法。

在实际的软件测试过程中，研究者专注于度量和建模那些对软件失效和故障检测排除至关重要的核心数据。这种方法论的采用，使得研究者能够准确捕捉并分析软件可靠性增长的长期趋势。在国内外的大型软件生命周期管理实践中，这种以核心要素为焦点的分析方法已经成为软件工程研究领域的标准范式。通过这种范式，研究者能够更有效地识别和解决软件工程中的复杂问题，从而推动软件可靠性的持续提升和优化。

4　模型性能影响评测

4.1　模型选择

本节将前文中提出的模型在真实的失效数据集上进行实验验证，用于分析 $a(t)$ 对模型的影响。这些数据集均来自于真实的工程，由不同的公司对外发布。

故障检测率 FDR；$b(t)$ 用于描述当前测试环境下故障被检测出的程度大小，故障修复概率 $p(t)$ 表示当前修复环境下故障被修复的概率，二者均与整体测试策略（技术、工具、测试人员技能等）、修复策略等紧密相关，可存在多种函数形式。这里主要关注 $a(t)$ 对模型的影响，不失一般性，不妨设定 $b(t)=b$ 常量，$p(t)=p$ 常量。不完美排错就是不断放宽传统完美排错中假设的条件，使之考虑的实际因素越来越靠近真实情况。鉴于不完美排错更加真实地描述了实际的测试过程，因此本文拟选择 7 个典型的不完美排错相关的 SRGMs（如表 3 所列）在 12 个数据集（DS_1-DS_6[73-78]，DS_7[74]，DS_8-DS_9[79-80]，$DS_{10}-DS_{12}$[81]）上进行验证。尽管现代软件开发方法、语言和流程发生了很多变化，但在实际的软件测试过程中，大多数的现代软件仍遵循与以前的开源软件测试相同或相似的规则。因此，本实验采用了经典故障数据集，并结合近年来新的数据集来评估和验证模型的性能。这不仅对现代软件测试具有实际的参考价值和指导意义，还有助于深入分析在不同类型故障总数下，各个模型之间的本质差异。

表 3　$a(t)$ 视角下典型不完美排错相关的 SRGMs

Table 3　SRGMs related to typical imperfect troubleshooting from the perspective of $a(t)$

模型	类型	累积故障检测数量 $m(t)$ 与其他参变量设置解释
M-1：Y-Lin[50]	Concave+ID	$m(t)=a(1-\mathrm{e}^{-bt})(1-\frac{\alpha}{b})+\alpha at$；$a(t)=a(1+\alpha t)$，$\alpha$ 为故障引入率；$b(t)=b$
M-2：P-N-Z[20]	S-shaped+Concave+ID	$m(t)=\frac{a[(1-\mathrm{e}^{-bt})(1-\frac{\alpha}{b})+\alpha t]}{1+\beta\mathrm{e}^{-bt}}$；$a(t)=a(1+\alpha t)$；$b(t)=\frac{b}{1+\beta\mathrm{e}^{-bt}}$；$a(t)$ 为 t 的线性增函数；$b(t)$ 为弯曲 S 型函数，且非降
M-3：P-Z[48]	ID	$m(t)=\frac{(c+a)(1-\mathrm{e}^{-bt})-\frac{ab}{b-\alpha}(\mathrm{e}^{-\alpha t}-\mathrm{e}^{-bt})}{1+\beta\mathrm{e}^{-bt}}$；$a(t)=c+a(1-\mathrm{e}^{-\alpha t})$；$b(t)=\frac{b}{1+\beta\mathrm{e}^{-bt}}$
M-4：P-Z[48]	ID	$m(t)=a(1+\beta t)[1-\mathrm{e}^{-\frac{b^2t^2}{2(1+bt)}}]$；$a(t)=a(1+\beta t)$
M-5：Pham[20]	ID	$m(t)=\frac{a}{1+\sigma\mathrm{e}^{-b(1+\sigma)t}}\left[(1-\mathrm{e}^{-b(1+\sigma)t})\left(1-\frac{r}{b(1+\sigma)}\right)+rt\right]$；$a(t)=a(1+rt)$；$b(t)=\frac{b(1+\sigma)}{1+\sigma\mathrm{e}^{-b(1+\sigma)t}}$，$\sigma$ 为测试过程学习因子，可正可负
M-6：Ohba-Chou[82]	ID	$m(t)=\frac{a}{1-r}[1-\mathrm{e}^{-(1-r)bt}]$；$b(t)=b$；$a(t)=\frac{a}{1-r}[1-r\mathrm{e}^{-(1-r)bt}]$
M-7：Wang-Zhang[81]	ID	$m(t)=\frac{a\mathrm{e}^{1-\mathrm{e}^{-\theta t^d}+\frac{1}{2}\sigma^2t}}{1+\beta\mathrm{e}^{-bt}}-\frac{a\sigma^2\mathrm{e}^{-bt}(\mathrm{e}^{(b+\frac{1}{2}\sigma^2)t}+\frac{2b+\sigma^2}{\sigma^2}-1)}{(1+\beta\mathrm{e}^{-bt})(2b+\sigma^2)}$；$b(t)=\frac{b}{1+\beta\mathrm{e}^{-bt}}$，$\sigma$ 表示不规则波动的大小并且是正的常数值，其中 θ 和 d 分别表示故障引入强度的速率参数和形状参数

不失一般性，在实验中统一将式（6）－式（10）中的故障检测率 $b(t)$ 设置为常量 b，故障排除率 $p(t)$ 设置为常量 p，重点关注不同总故障数量 $a(t)$ 在框架模型中的性能表现。带入常量后的框架模型所对应的累积故障检测数量函数表达式如表 4 所列。

对上述不完美排错相关的 SRGM 在公开发表的多个真

实计算机工程系统失效数据集上进行实验，观测不同模型的性能并进行比较。另一方面，从拟合与预测角度分析失效数据集与 SRGM 的关系及影响，并对 SRGM 中的参函数进行剖析，涵盖以下几者间的影响：FDS（Failure Data Set）↔$m(t)$，FDS↔$a(t)$，$a(t)$↔$R(t)$，以及 $m(t)$↔$a(t)$。最后，从发布方与科研人员视角对当前失效数据集的不足进行分析，并据此给出软件最优发布与失效数据集发布的建议，为后续 SRGM 的决策研究进行铺垫。

表 4 软件总故障数量函数 $a(t)$在框架模型中对应的 $m(t)$

Table 4 Corresponding $m(t)$ of the total software failure quantity function $a(t)$ in the framework models

模型	$a(t)$形式与分类	累积故障检测数量 $m(t)$
M-8	乐观类型 $a(t)=c+a(1-e^{-\alpha t})$	$m(t)=\frac{(a+c)(1-e^{-bpt})}{p}+\frac{ab(e^{-\alpha t}-e^{-bpt})}{\alpha-bp}$
M-9	悲观类型 $a(t)=ae^{\alpha t}$	$m(t)=\frac{ab(e^{\alpha t}-e^{-bpt})}{\alpha+bp}$
M-10	悲观类型 $a(t)=a(1+\alpha t)$	$m(t)=\frac{a(1+\alpha t-e^{-bpt})}{p}-\frac{a\alpha(1-e^{-bpt})}{bp^2}$
M-11	悲观类型 $a(t)=a(1+rt)^2$	$m(t)=\frac{a(p^2(rt+1)^2b^2-2pbr(rt+1)+2r^2-(p^2b^2-2pbr+2r^2)e^{-bpt})}{b^2p^3}$
M-12	折中类型 $a(t)=a+\alpha m(t)$	$m(t)=\frac{a(1-e^{(\alpha b-bp)t})}{p-\alpha}$
M-13	间接求解类型 $\frac{da(t)}{dt}=\beta(t)\cdot\frac{d(p(t)\cdot m(t))}{dt}$	$m(t)=\frac{a}{p(\beta-1)}(e^{pb(\beta-1)t}-1)$

4.2 拟合性能实验与分析

由于页面限制，本节绘制了 12 个公开发表的失效数据集的累积检测故障数量曲线与本文参与比较的 SRGM 模型估计值曲线（可联系作者获得更多实验信息），如图 4 所示。其中模型的拟合曲线越接近于真实的失效数据曲线，表明拟合效果越好。

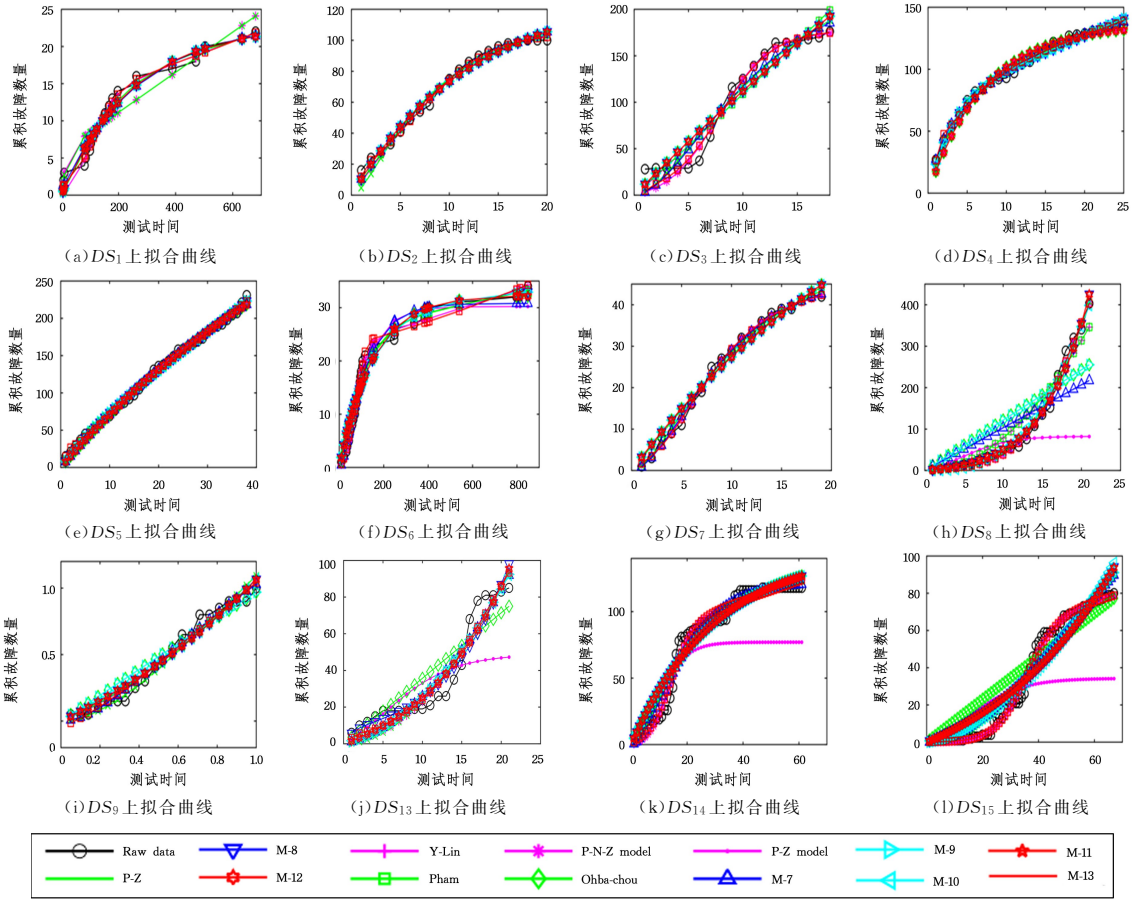


图 4 参与对比分析的模型在 DS_1-DS_{12} 的拟合曲线

Fig. 4 Fitting curves of models involved in comparative analysis in DS_1-DS_{12}

通过比较已有模型（Y-Lin，Ohba-Chou 等）与提出的框架模型（M-8－M-12 和 M-13）对应的拟合曲线能验证提出的框架模型的合理性。另外，由于模型 M-8－M-12 均是在统一的框架模型下固定故障检测率 $b(t)=b$ 和故障排除率 $p(t)=p$，通过带入不同软件总故障数量函数推导而来的，故分析模型 M-8－M-12 在各种数据集上的表现可以估计 $a(t)$性能的优劣。从不同数据集上的实验拟合曲线结果可以得出以下结论。

1）整体上，除了 M-3 在 $DS_{8,12,14,15}$ 上的拟合曲线与失效

数据集曲线出现严重偏差外，其他模型的拟合结果与失效数据集保持一致。这说明提出的基于 NHPP 假设的两个框架模型——直接设定总故障函数与微分求解总故障函数具有可行性，带入不同形式的总故障函数 $a(t)$ 与故障检测率函数 $b(t)$ 后能够对软件测试过程检测出的故障数量进行估计。

2）细节上，M-2 在 DS_2，DS_3，DS_5，DS_6，DS_7，DS_8，DS_{10}，DS_{11}，DS_{12} 这 9 个失效数据集上的拟合曲线最接近失效数据集曲线，具有最好的拟合性能。究其原因：一方面 M-2 模型采用了线性增长的总故障数量函数，另一方面模型的故障检测率 $b(t)$ 设定为更为灵活的 S 型形式 $b(t)=b/(1+\beta e^{-bt})$。这说明除了软件总故障数量函数以外，不同形式的故障检测率函数 $b(t)$ 也会对 SRGM 模型的性能具有重要影响。在固定故障检测率函数的框架模型 M-8－M-12 中，乐观类型的总故障数量函数对应的 M-8 模型的拟合性能最好，在 DS_1，DS_3，DS_4，DS_6，DS_7，DS_9，DS_{10}，DS_{11}，DS_{12} 这 9 个数据集上模型的估计曲线与数据集失效曲线最为接近，说明在以上的数据集上，总故障数量存在上限的乐观类型能够更好地描述软件测试过程中的总故障数量变化趋势。悲观类型的软件总故障数量对应的模型 M-9－M-11 表现出较为一致的结果，在特定数据集上的拟合性能表现优秀（例如 DS_5），但在绝大多数数据集上的表现不佳。其中悲观类型——线性增长的总故障函数对应的 M-9 模型在 DS_7 上拟合性能较好，其性能表现优于另外两个悲观类型对应的模型。折中类型与间接求解类型的 $a(t)$ 对应的模型具有近似的性能表现，在大部分数据集上的拟合曲线都较为接近真实数据集曲线。

3）在 S 型增长的 DS_3 上，除了 S 型 $b(t)$ 的 M-2 与 M-3 能够较好地拟合出 S 型增长数据集故障曲线所表现的在软件测试周期中故障检测率由低到高、达到峰值后逐渐降低的真实测试情况，其他模型的拟合曲线均表现为直线，并不能准确地描述失效数据集的增长趋势。特别是故障检测率 $b(t)$ 为常量，乐观类型、悲观类型以及折中类型的总故障数量对应的 M-8－M-12 都无法拟合 S 型增长的累计检测故障数量。这一现象可解释为，总故障数量函数 $a(t)$ 作为软件可靠性增长模型中的重要影响因素之一，其对模型的性能表现具有一定局限性。对于增长形式更为复杂的 S 型增长的失效数据集，可以在框架模型中使用更为灵活的 S 型故障检测率函数，以提升模型的性能。

4）本文参与比较的 13 个 SRGM 模型中，并没有某一个模型在所有数据集上都表现良好，例如 M-2 在 8 个数据集上表现出色，但在 DS_3，DS_6 这两个数据集上的拟合曲线表现很差；又如 M-3 在 DS_{10}，DS_{11}，DS_{12} 这 3 个数据集上拟合较差。究其原因，不同软件系统的架构不尽相同，不同软件的测试环境也具有较大差异，这些差别最终在失效数据集上表现出来。而软件可靠性增长模型是基于抽象假设下通过一定数学方法对累积检测故障数量进行估计的方法，这必然无法覆盖情况复杂的所有失效数据集。

4.3　预测性能实验与分析

为了观测模型的预测性能，本节绘制了参与比较模型的预测 RE 曲线，如图 5 所示。

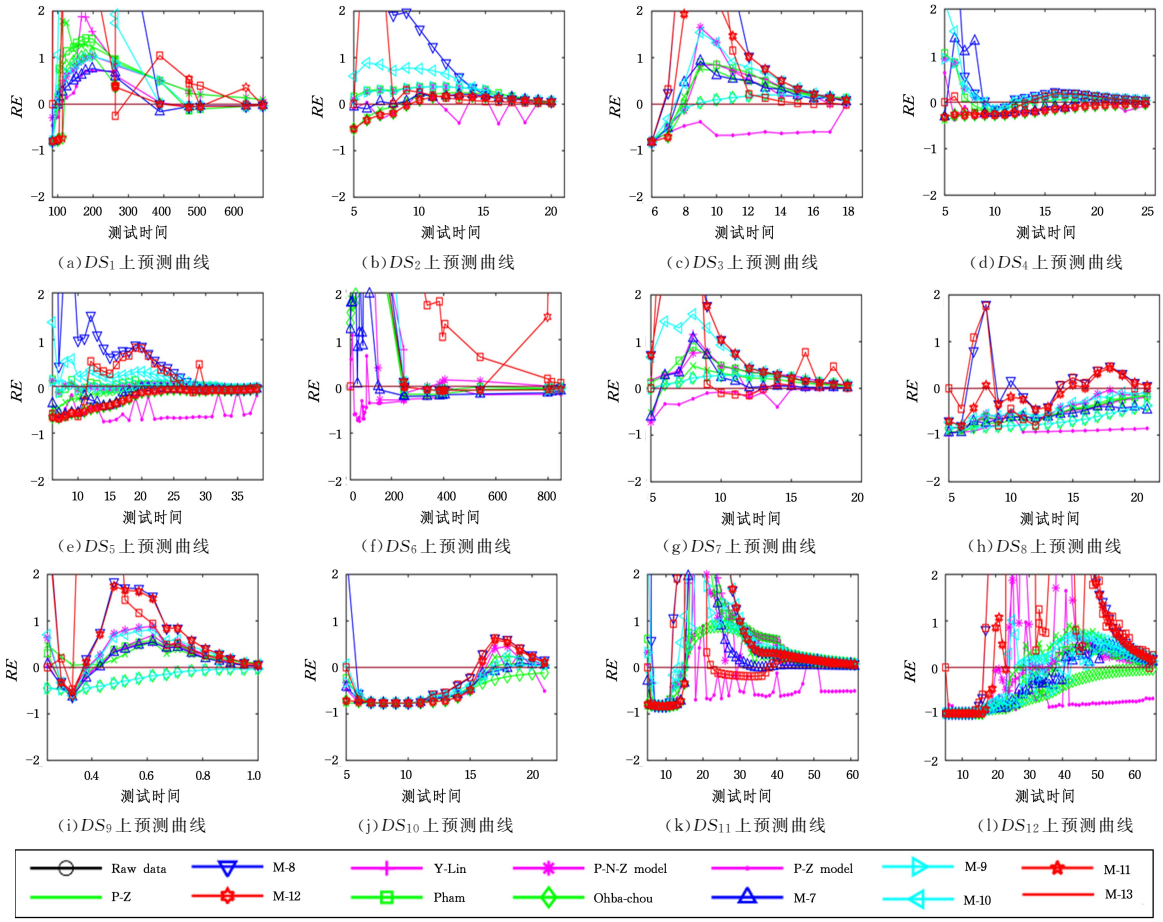


图 5　参与对比分析的模型在 DS_1－DS_{12} 的预测曲线

Fig. 5　Prediction curves of models involved in comparative analysis from DS_1－ DS_{12}

RE 曲线越趋近于 0，表明预测性能越好，其中位于 0 以上是正向预测，位于 0 以下是负向预测。基于数据集的预测可以看作是模型对未来测试性能的描述能力，也反映出模型在后续时刻累积检测出故障的准确程度。

分析以上模型的预测曲线，可以得出以下结论。

1)整体上，除了 M-2 在部分数据集上的拟合曲线存在较大偏差且不收敛于零水平线之外，绝大多数模型都表现出良好的预测性能。并且随着测试时间的增加，收集到的软件故障相关数据越多，模型的预测性能越好。

2)模型的预测性能与拟合性能具有一定的相关性。折中类型与微分求解形式的总故障数量函数对应模型的预测曲线较为接近，在大多数据集上表现较好，具有较为稳定的预测性能。乐观类型与线性悲观类型的 $a(t)$ 对应的模型预测能力具有波动性，在部分数据集上性能表现出色(如 DS_5，DS_{11} 等)，但在部分数据集上也存在一定偏差。指数悲观类型与二次悲观类型的 $a(t)$ 对应的模型的预测能力较为不稳定，在部分数据集上的预测性能表现不佳。

3)对于 S 型增长数据集 DS_2，虽然框架模型 M-8－M-13 的拟合曲线无法展现出数据集 S 型增长的特殊趋势，模型的拟合性能表现不佳，但其对应的预测曲线在软件测试前半程偏差较大，而在软件测试后期，一系列模型的预测曲线仍趋近于零水平线。这表明若软件可靠性增长模型能够基本对失效数据集的增长趋势大致拟合，在收集到一定数量测试数据的情况下，模型依然可以对软件的故障数量进行较好的预测。

4.4 相关讨论

基于上述 13 个模型在 15 个公开发表的失效数据集上的实验结果，可进行如下深入讨论。

1)因为软件可靠性增长模型的建模假设与真实软件测试过程存在差异，所以并不存在一个模型可以良好拟合所有失效数据集。提出的模型假设考虑越多、越复杂的现实因素，则模型的拟合预测性能越好。相较于固定为常量 a 的总故障数量，考虑不完美排错的随测试时间 t 变化的总故障数量函数 $a(t)$ 对应的可靠性模型具有更好的性能表现。

2)综合实验结果，可对本文中参与统一框架模型对比的不同类型的总故障数量函数的性能表现做出初步排序：乐观类型＞线性悲观类型＞二次悲观类型＞指数悲观类型＞折中类型≈微分求解类型。对总故障数量持有保守假设的模型具有更好的拟合性能表现，例如乐观类型与线性悲观类型。而软件故障随测试时间 t 快速增长的二次类型与指数类型在个别数据集上的拟合性能良好。折中类型与求解类型在拟合性能方面表现折中，在大多数数据集上的预测性能稳定。

3)软件总故障函数是对软件系统的全部故障数量的描述，既包括已检测故障，也包括尚未发现的故障数量。在软件可靠性建模时，可人为地将总故障数量设定为有增长上限的乐观类型、无增长上限的悲观类型、随累积检测故障数量变化的折中类型与微分求解类型。在同一个数据集上，乐观与悲观类型总故障数量对应的模型具有截然相反的性能表现，若乐观类型对应模型具有最好的性能表现，则悲观类型对应模型的性能较差，反之亦然。根据软件总故障函数定义可推断在乐观类型表现良好的失效数据集中，数据集对应的软件系统的总故障数量存在上限，随着测试时间的增加，最终总故障数量也保持稳定；悲观类型表现出色的数据集对应的软件系统没有总故障数量上限，随测试时间的增加，软件系统的故障数量逐渐增加。

基于以上分析，本文参与实验的大多失效数据集都是软件总故障存在上限的乐观类型。但对软件系统的总故障数量的判断，也要结合软件编写人员的技能水平、软件测试人员的熟练程度、软件开发的规范程度等实际因素进行综合判断。

4)不同类型总故障数量函数为了提高软件可靠性增长模型的性能表现，除了考虑软件总故障数量这一因素外，还应确定合适的故障检测率 FDR、故障排除概率 $p(t)$、测试工作量 TE(testing effort)、测试覆盖率 TC(testing coverage)等因素，使得建模过程更贴近真实软件测试情况，发挥出内在融合的综合效应，以便获得具有更好拟合预测性能表现的软件可靠性增长模型。

5 研究挑战与亟待解决的问题

5.1 研究挑战

5.1.1 建立能够描述软件生命周期中各阶段的故障总数函数

在软件的需求分析阶段，需求分析师可能因对业务逻辑认识不够而造成“画像不准”，带来质量问题；在软件设计阶段，设计者(如架构师)可能因设计缺陷而带来潜在的故障；在软件开发阶段，程序员开发能力的不足，致使软件中存在 Bug；在软件的测试与修改阶段，排错过程的整体策略与人员水平的差异会使得软件中被引入新的错误。这些缺陷、风险、不足和隐患，会使得软件在长期运行中出现各种故障甚至失效，严重时会带来巨大损失。如何建立覆盖软件主要生命周期的故障总数模型，比较准确地刻画故障总数的生成方式，定量地描述阶段性故障总数模型对可靠性的提升作用，是推动软件工程高质量发展的重要内容。限于软件生命周期各阶段流程建模，特别是理论模型的研究进展，以及当前软件测试发布信息的不完备等不足，突破建立多阶段软件故障总数模型的难度较大，这也成为整个可靠性研究中的一个挑战。

5.1.2 建立软件中故障总数与更多测试过程中因素的定量关系

软件可靠性是高可信软件的主要衡量指标之一，在软件的质量体系中占据重要地位。实际上，可靠性是由包括故障总数等在内的多因素综合作用的结果，单纯从某个因素进行研究或施策对可靠性的提升作用有限。从故障总数的角度来看，测试策略制定、测试资源分配、测试用例生成、测试路径选择等都会对故障排错和测试质量带来影响，只有建立多要素融合关联的定量关系模型才能更深入、全面和准确地研究可靠性的变化。在今后的可靠性理论研究中，应该重视这种多要素的综合性联系研究，科学看待可靠性的制约因素，合理地建立支撑可靠性随时间持续增长的数学模型，进而指导软件可靠性工程高质量的发展。

5.2 亟待解决的问题

5.2.1 在真实的软件工程开发中选择合适的故障总数模型

从本文的研究中可以看出，软件中故障总数对软件可靠

性有着重要影响，且软件中总故障个数的函数存在多种形式，因此在真实的软件开发中，如何确定或者制定合适的软件中总故障函数形式，对于指导软件测试、提高可靠性和及时发布软件具有重要意义。由于软件开发的不同阶段具有不同的特征，因此可以采用分阶段来确定合适的软件总故障个数函数的策略，这将有益于指导软件工程健康发展。从当前的研究来看，软件中故障总数函数还是以人为主观设定为主，缺乏足够的理论基础，尚难以对大型软件项目开发提供科学的指导依据，这一问题亟待突破。

5.2.2　基于故障总数建立更为精准的软件发布策略

软件发布是软件由开发阶段过渡到应用阶段的转折，对于决定软件质量、控制开发与测试成本、及时投入市场或交付给用户等具有重要意义。软件中故障总数是决定软件发布的重要因素，因此基于软件故障总数模型制定合理的发布策略是重要研究主题。软件发布并不是意味着软件中没有故障，而是故障不影响软件的使用（或影响范围与程度可接受），并经过使用反馈进行修改迭代提升再投入应用。通过对软件测试资源分配、成本控制与发布问题进行深入研究[83]，发现到目前为目，软件发布策略中对故障总数的考虑并不够，忽视了故障总数对可靠性的影响，使得软件发布不够精准。这里，从故障总数的角度提出软件发布的最优化决策模型，如式(20)所示：

$$\begin{cases} \mathrm{Min}[a(t)-c(t)] \\ \text{s.t. } R(x\mid T)\geqslant R_0, \text{where } 0<R_0<1 \text{ and } x>0 \\ \quad E[C(T)]\leqslant C_0 \end{cases} \tag{20}$$

其中，$c(t)$表示 t 时刻累积修复的故障数量，$C(T)$表示截止软件发布 T 时所消耗的成本。该最优化决策模型可以理解为：在成本不超支和可靠性达到最低限度的情况下，使得软件中剩余的故障数量最小。对该最优化数学模型进行求解，即可确定软件最优发布时间 T。

5.2.3　基于更加丰富的失效数据集发布信息建立故障总数模型

因为测试与排错的过程中存在引入新故障的可能，所以 $a(t)$应随 t 有所增加。对此，绘制出了不同 $a(t)$模型在部分失效数据集上的变动曲线，如图 6 所示。

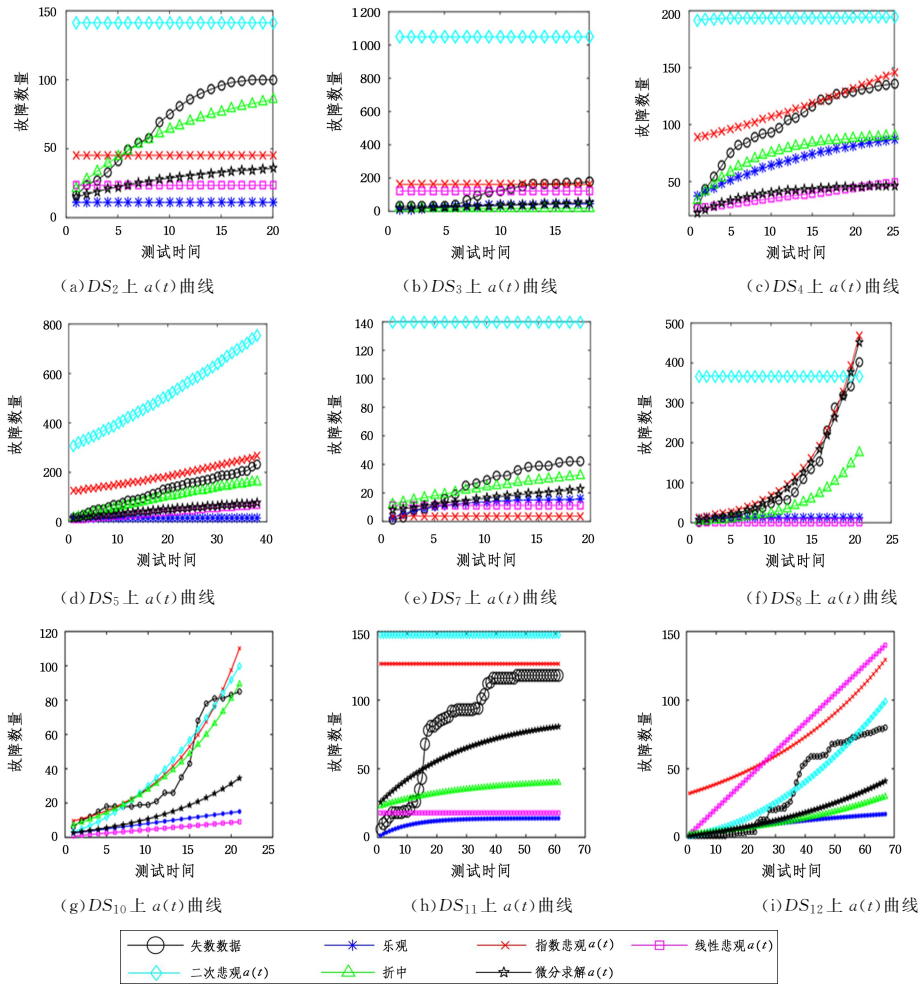


图 6　部分失效数据集失效数据曲线与通过模型估计的总故障曲线

Fig. 6　Partial failure data set failure data curves and total failure curves estimated by models

人为设定 $a(t)$的差异，使得曲线走势既有线性增加，又有指数以及更加复杂的增加变化趋势。失效数据集记录的是软件总故障数量的一部分，二者理应类似，但观察图 5 发现二者存在较大差距，这是由于软件可靠性建模人员对 $a(t)$的假设存在偏差。由于 FDS 发布方是测试过程的主导者，因此其具备获得测试过程中更多信息的能力，但当前的 FDS 中均没有此方面的信息。这为故障总数模型研究和可靠性研究带来了严重制约，成为需要突破的问题。我们呼吁 FDS 发布方（公

司或研究所)发布更为全面的失效数据集的信息,帮助软件可靠性建模人员对故障总数量进行更为精准的建模。

5.2.4 软件结构与形态发生了显著变化情况下故障总数的演变特征

软件的开发方式、技术、结构、形态等都会给软件中故障的分布与数量的增长带来影响。当前研究仍然是将软件作为一种形态来进行统一看待,没有区分软件自身的变化特征,忽视了软件自身发展引发故障总数的变化。例如,随着开发模式(开源式开发、迭代式开发、敏捷开发、众筹开发等)与开发技术的发展,特别是需求提升与应用创新,构件软件、网构软件、开源软件、平台软件、社交与通信软件、移动端软件等已成为软件中的主流,针对这些软件形态与结构的变化来研究软件中故障的分布与数量增长情况应得到足够重视。故障往往是由开发中的问题或不足引起的,因此要深入结合软件架构与开发特征来研究故障总数模型。例如,应该将软件架构模型、开发范式、软件开发与测试的理论技术等与故障总数分布及生成等联系起来,建立考虑更多软件开发特征的模型,以提高研究的适配性、精准性和有效性。

结束语 软件中故障总数是软件可靠性建模、评测、增长的重要组成元素,对于测试资源分配、成本管控和最优发布也具有重要的影响。软件中故障总数既难以确定又难以全部排除,当前软件可靠性研究中多以直接设定故障总数为主,这为建立更为准确的可靠性模型和确定更为准确的测试资源投入带来了一定困难。依据软件自身结构等特征,建立能够描述软件中故障数量增长的模型,不仅对确定排错中引入新故障带来直接益处,也成为推动可靠性工程发展的重要力量,是可靠性研究领域值得探索的核心主题。

本文对软件中故障总数进行了全面述评,包括区分不同的函数模型,进而从软件中故障总数的角度建立两种不完美排错情况的软件测试模型和多种分类子模型。针对得到的可靠性模型,从拟合与预测角度研判分析了模型性能间的差异,剖析了故障总数对可靠性的影响。期望我们的工作可以为推动故障总数研究、可靠性研究提供有价值的参考或借鉴,为在经济与社会发展中发挥越来越重要的软件的可靠性研究和应用做出有益的贡献。

参 考 文 献

[1] ZHANG J, LU Y, ZHANG B H, et al. Reliability analysis algebraic approach to software evolution[J]. Acta Automatica Sinica, 2021, 47(1): 148-160.

[2] Academician Forum | MEI HONG academician: entering the era of software definition[EB/OL]. https://m.thepaper.cn/baijiahao_6956608.

[3] MEI H. Future world defined by software[J]. Satellite & Network, 2018(6): 28-33.

[4] MEI H. Everything can be interconnected and everything can be programmed[J]. Fangyuan Magazine, 2018(12): 58-59.

[5] XU Y, SUN L F, ZHANG T H R, et al. Examining the Quality of Bug Report Titles: An Empirical Study [J]. Computer Science, 2023, 50(12): 14-23.

[6] JIANG J J, CHEN J J, XIONG Y F. Survey of automatic program repair techniques[J]. Journal of Software, 2021, 32(9): 2665-2690.

[7] CAO X D, HUANG Z Q, CHEN Y J, et al. An Overview of Research on Vulnerability Database Construction and Application [J]. Chinese Journal of Computers, 2024, 47(5): 1082-1119.

[8] SU XH, ZHENG W N, JIANG Y, et al. Research and Progress on Learning-Based Source Code Vulnerability Detection[J]. Chinese Journal of Computers, 2024, 47(2): 337-374.

[9] FUJ M, JIANG Y Q, HE J, et al. Cryptocurrency Mining Malware Detection Method Based on Sample Embedding[J]. Computer Science, 2024, 51(1): 327-334.

[10] GU J X, SUN H Y, HAN D, et al. Software security vulnerability mining based on deep learning[J]. Journal of Computer Research and Development, 2021, 58(10): 2140-2162.

[11] ZHANG F, XU M D, ZHAO H J, et al. Real-time trust measurement of software: behavior trust analysis approach based on noninterference[J]. Journal of Software, 2019, 30(8): 2268-2286.

[12] YADAV S S, KUMAR A, JOHRI P, et al. Testing effort-dependent software reliability growth model using time lag functions under distributed environment[M]// System Assurances. 2022: 85-102.

[13] PRADHAN V, KUMAR A, DHAR J. Modelling software reliability growth through generalized inflection S-shaped fault reduction factor and optimal release time[J]. Journal of Risk and Reliability, 2022, 236. (1): 18-36.

[14] DUAN W X, HU M, ZHOU Q, et al. Reliability in cloud computing system: a review[J]. Journal of Computer Research and Development, 2020, 57(1): 102-123.

[15] WU Y J, CHEN H B, BAO Y G, et al. Preface to the topic of Frontier progress of system software[J]. Journal of Software, 2020, 31(10): 2981-2982.

[16] WANG J, ZHANG C. Software reliability prediction using a deep learning model based on the RNN encoder-decoder[J]. Reliability Engineering & System Safety, 2018(170): 73-82.

[17] AGGARWAL A G, GANDHI N, VERMA V, et al. Multi-Release software reliability growth assessment: An approach incorporating fault reduction factor and imperfect debugging[J]. International Journal Mathematics in Operational Research, 2019, 15(4): 446-463.

[18] SONG K Y, CHANG I H, PHAM H. A three-parameter fault-detection software reliability model with the uncertainty of operating environments[J]. Journal of Systems Science and Systems Engineering, 2017, 26(1): 121-132.

[19] KAPUR P K, SINGH V B, ANAND S, et al. Software reliability growth model with change-point and effort control using a power function of the testing time[J]. International Journal of Production Research, 2008, 46(3): 771-787.

[20] PHAM H, NORDMANN L, ZHANG Z. A general imperfect-software-debugging model with S-shaped fault-detection rate [J]. IEEE Transactions on Reliability, 1999, 48(2): 169-175.

[21] NAGARAJU V, WANDJI T, FIONDELLA L. Improved algorithm for non-homogeneouspoisson process software reliability growth models incorporating testing-effort [J]. International

Journal of Performability Engineering. 2019,15(5):1265-1272.

[22] PENG R,MA X Y,ZHAI Q Q,et al. Software reliability growth model considering first-step and second-step fault dependency [J]. Journal of Shanghai Jiaotong University(Science),2019,24:477-479.

[23] SARAF I,LQBAL J. Generalized multi-release modelling of software reliability growth models from the perspective of two types of imperfect debugging and change point[J]. Quality & Reliability Engineering International. 2019,35(7):2358-2370.

[24] SARAF I,LQBAL J. Generalized software fault detection and correction modeling framework through imperfect debugging, error generation and change point[J]. International Journal of Information Technology,2019,11(4):751-757.

[25] ERTO P,GIORGIO M,LEPORE A. The generalized inflection S-shaped software reliability growth model[J]. IEEE Transactions on Reliability,2020,69(1):228-244.

[26] LEE T Q,YEH C W,FANG C C. Bayesian software reliability prediction based on yamada delayed s-shaped model[J]. Applied Mechanics and Materials,2014,490:1267-1278.

[27] HAQUE M A,AHMAD N. An effective software reliability growth model[J]. Safety and Reliability,2021(2):1-12.

[28] MUNDE A. An empirical validation for predicting bugs andthe release time of open source software using entropy measures—Software reliability growth models[M]// System Assurances. 2022:41-49.

[29] ZHU M,PHAM H. A generalized multiple environmental factors software reliability model with stochastic fault detection process[J]. Annals of Operations Research,2022,311:525-546.

[30] MINAMINO Y,INOUE S,YAMADA S. Extension of software reliability growth models by several testing-time functions [M]// System Assurances. 2022:155-174.

[31] AHMAD N,BOKHARI M U,QUADRI S M K,et al. The exponentiated Weibull software reliability growth model with various testing-efforts and optimal release policy: a performance analysis[J]. International Journal of Quality & Reliability Management,2008,25(2):211-235.

[32] MANJULA T,JAIN M,GULATI T R. Cost optimization of a software reliability growth model with imperfect debugging and a fault reduction factor[C]// Proceedings Engineering Mathematics and Applications Conference. 2014:182-196.

[33] LI H F,LI Q Y,LU M Y. Software reliability modeling with logistic test coverage funtion[J]. Journal of Computer Research and Development,2011,48(2):232-240.

[34] LI H F,WANG S Q,LIU C,et al. Software reliability model considering both testing effort and testing coverage[J]. Journal of Software,2013(4):749-760.

[35] WANG J Y,ZHANG C,MI X P,et al. Software reliability growth model based on weibull distribution introduced faults [J]. Journal of Software,2019,30(6):1759-1777.

[36] ZHANG C,YI W M,BAI R,et al. Utility and verification of failure data set in SRGM[J]. Computer Engineering & Science, 2020,42(6):1012-1020.

[37] GOEL L,OKUMOTO K. Time-dependent error-detection rate model for software reliability and other performance measures [J]. IEEE Transactions on Reliability,1979,R-28(3):206-211.

[38] YAMADA S,HISHITANI J,OSAKI S. Software-reliability growth with a Weibull test-effort: a model and application[J]. IEEE Transactions on Reliability,1993,42(1):100-106.

[39] PHAM T,PHAM H. A generalized software reliability model with stochastic fault-detection rate[J]. Annals of Operations Research,2019,277(1):83-93.

[40] WANG J,MI X. Open Source Software reliability model with the decreasing trend of fault detection rate[J]. The Computer Journal,2019,62(9):1301-1312.

[41] WANG J Y,WU Z B,SHU Y J,et al. A Software reliability model with irregular changes of fault detection rate[J]. Journal of Software,2015,26(10):2465-2484.

[42] ZHANG C,LIU H W,BAI R,et al. Review on fault detection rate in reliability model[J]. Journal of Software,2020,31(9): 2802-2825.

[43] ZHANG C,MENG F C,KAO Y G,et al. Survey of software reliability growth model[J]. Journal of Software,2017,28(9): 2402-2430.

[44] ZHANG C,MENG F C,WAN K,et al. Analysis on SRGM modeling categories and performances[J]. Journal of Harbin Institute of Technology,2016,48(8):171-178.

[45] ZHANG C,LYU W G,QIU Z Y,et al. Testing coverage software reliability model under imperfect debugging[J]. Journal of Hunan University(Natural Sciences),2021,48(4):26-35.

[46] YAMADA S,OHBA M,OSAKI S. S-shaped reliability growth modeling for software error detection[J]. IEEE Transactions on Reliability,1983,32(5):475-484.

[47] HUANG C,LYU M,KUO S. A unified scheme of somenonhomogenous poisson process models for software reliability estimation[J]. IEEE Transactions on Software Engineering,2003, 29(3):261-269.

[48] PHAM H,ZHANG X. An NHPP software reliability model and its comparison[J]. International Journal of Reliability, Quality and Safety Engineering,1997,4(3):269-282.

[49] PHAM H. System software reliability[M]. London: Springer, 2007.

[50] YAMADA S,TOKUNO K,OSAKI S. Imperfect debugging models with fault introduction rate for software reliability assessment[J]. International Journal of Systems Science,1992, 23(12):2241-2252.

[51] PHAM H,ZHANG X. NHPP software reliability and cost models with testing coverage[J]. European Journal of Operational Research,2003,145(2):443-454.

[52] PHAM H. An imperfect-debugging fault-detection dependent-parameter software[J]. International Journal of Automation and Computing,2007,4(4):325-328.

[53] AHMAD N,KHAN M,RAFI L. A study of testing-effort dependent inflection s-shaped software reliability growth models with imperfect debugging[J]. International Journal of Quality & Reliability Management,2010,27(1):89-110.

[54] HUANG C,KUO S,LYU M. An assessment of testing-effort dependent software reliability growth models[J]. IEEE Transactions on Reliability,2007,56(2):198-211.

[55] PENG R, HU Q P, NG S H, et al. Testing effort dependent software FDP and FCP models with consideration of imperfect debugging [C]// 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement. IEEE, 2010: 141-146.

[56] ZHANG X, TENG X, PHAM H. Considering fault removal efficiency in software reliability assessment[J]. IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans, 2003, 33(1): 114-120.

[57] SAMEERA M S, KANCHARLA G R, PRASAD R S. Software reliability measurement using combined Goel OKUMOTO and ANOM perfect debugging model[J]. Journal of Advanced Research in Dynamical and Control Systems, 2019, 11: 780-787.

[58] ZHANG C, CUI G, MENG F C, et al. A study of optimal release policy for SRGM with imperfect debugging[J]. Journal of Engineering Science and Technology Review, 2013, 6(3): 111-118.

[59] LIN C T. Analyzing the effect of imperfect debugging on software fault detection and correction process via a simulation framework[J]. Mathematical and Computer Modelling, 2011, 54: 3046-3064.

[60] JAIN M, MANJULA T, GULATI T R. Imperfect debugging study of SRGM with fault reduction factor and multiple change point[J]. International Journal of Mathematics in Operational Research, 2014, 6(2): 155-175.

[61] PENG R, LI Y F, ZHANG W J, et al. Testing effort dependent software reliability model for imperfect debugging process considering both detection and correction[J]. Reliability Engineering & System Safety, 2014, 126: 37-43.

[62] ZHANG C, CUI G, BIAN Y L, et al. Component-based software reliability process simulation considering imperfect debugging [J]. High Technology Letters, 2014, 20(1): 9-15.

[63] WANG J. An imperfect software debugging model considering irregular fluctuation of fault introduction rate[J]. Quality Engineering, 2017, 29(3): 377-394.

[64] WANG J, WU Z. Study of the nonlinear imperfect software debugging model[J]. Reliability Engineering & System Safety, 2016, 153: 180-192.

[65] WANG J, WU Z, SHU Y, et al. An imperfect software debugging model considering log-logistic distribution fault content function[J]. Journal of Systems & Software, 2015, 100: 167-181.

[66] KAPUR P K, PHAM H, ANAND S, et al. A unified approach for developing software reliability growth models in the presence of imperfect debugging and error generation[J]. IEEE Transactions on Reliability, 2011, 60(1): 331-340.

[67] HUANG C Y, LYU M R. Optimal release time for software systems considering cost, testing-effort, and test efficiency [J]. IEEE Transactions on Reliability, 2005, 54(4): 583-591.

[68] HUANG C Y, LO J H. Optimal resource allocation for cost and reliability of modular software systems in the testing phase[J]. Journal of Systems and Software, 2006, 79(5): 653-664.

[69] HUANG C Y. Cost-reliability-optimal release policy for software reliability models incorporating improvements in testing efficiency[J]. Journal of Systems and Software, 2005, 77(2): 139-155.

[70] HUANG C Y, LYU M R. Estimation and analysis of some generalized multiple change-point software reliability models[J]. IEEE Transactions on Reliability, 2011, 60(2): 498-514.

[71] KUO S Y, HUANG C Y, LYU M R. Framework for modeling software reliability, using various testing-efforts and fault-detection rates[J]. IEEE Trans on Reliability, 2001, 50(3): 310-320.

[72] WANG J, WU Z, SHU Y, et al. An optimized method for software reliability model based on nonhomogeneous Poisson process[J]. Applied Mathematical Modelling, 2016, 40(13/14): 6324-6339.

[73] EHRLICH W, PRASANNA B, STAMPFEL J, et al. Determining the cost of a stop-test decision (softwarereliability) [J]. IEEE Software, 1993, 10(2): 33-42.

[74] WOOD A. Predicting software reliability [J]. Computer, 1996, 29(11): 69-77.

[75] STRINGFELLOW C, ANDREWS A A. An Empirical method for selecting software reliability growth models[J]. Empirical Software Engineering, 2002, 7(4): 319-343.

[76] ZHANG X, PHAM H. A software cost model with warranty cost, error removal times and risk costs[J]. IIE Transactions, 1998, 30(12): 1135-1142.

[77] MISRA P N. Software reliability analysis [J]. IBM Systems Journal, 1983, 22(3): 262-270.

[78] HOSSAIN S A, DAHIYA R C. Estimating the parameters of a non-homogeneous Poisson-process model for software reliability [J]. IEEE Transactions on Reliability, 1993, 42(4): 604-612.

[79] BAI C G, HU Q P, XIE M, et al. Software failure prediction based on a Markov Bayesian network model[J]. Journal of Systems and Software, 2005, 74(3): 275-282.

[80] ZHANG X, PHAM H. Software field failure rate prediction before software deployment[J]. Journal of Systems and Software, 2006, 79(3): 291-300.

[81] WANG J, ZHANG C. An Open-Source Software Reliability Model Considering Learning Factors and Stochastically Introduced Faults[J]. Applied Sciences, 2024, 14(2): 708-742.

[82] OHBA M, CHOU X M. Does imperfect debugging affect software reliability growth? [C]// Proceedings of the 11th International Conference on Software Engineering. ACM, 1989: 237-244.

[83] ZHANG C, CUI G, LIU H W, et al. Software test resources and cost control and optimal release policy[J]. Journal of Harbin Institute of Technology, 2014, 46(5): 51-58.

ZHANG Ce, born in 1978, Ph.D, professor, master supervisor, is a senior member of CCF (No. 12696S). His main research interests include reliability modeling and evaluation, security analysis in the Internet of Things, trusted computing and so on.

(责任编辑:李亚辉)