文章编号: 1003- 207(2010)06- 0089- 08

基于失信因子的软件缺陷预测模型

占济舟[1], 周献中[1], 赵佳宝[1], 王建峰[2]

(1. 南京大学工程管理学院, 江苏 南京 210093; 2. 南京大学数学系, 江苏 南京 210093)

摘 要: 软件缺陷是衡量软件项目可靠性的重要指标, 缺陷的预测技术也是软件质量度量的核心问题。在软件缺陷产生及生长过程机理分析的基础上, 将缺陷分为潜藏、被发现和被修复三种状态, 建立了在失信因子和约束因子的共同作用下, 缺陷状态动态转移的微分方程模型, 并对其平衡点的稳定性及失信因子的边界条件进行了分析。同时, 对 NASA 中一个软件项目进行实证分析, 利用软件测试及缺陷的修复记录对系统中潜藏的缺陷进行预测, 结果表明模型的合理性。

关键词: 缺陷; 失信因子; 微分方程; 预测

中图分类号: TP311.5 文献标识码: A

1 引言

软件作为一种依附于计算机硬件, 具有特定性质和功能的产物, 在人们的各种社会生活中得到了广泛应用。然而, 随着功能需求的不断增加, 软件系统逐渐庞大和复杂, 经常出现故障和失效, 与人们的预期发生偏离, 因而, 软件往往是不可信的。从软件失效的机理[1]来看, 缺陷的早期检测和及时修复可以有效地避免失效, 减少损失。软件缺陷的预测对于软件是否交付以及软件质量的评估都起着关键作用。

从 20 世纪 70 年代至今, 国内外学者在软件缺陷预测方面做了大量的研究工作。早期有 Halstead 模型和 McCabe 模型[2]; Fenton 提出的适用于生命周期中不同阶段的 Bayesian 模型[3]; Goel 和 Okumoto 的非齐次泊松过程(NHPP) 模型[4]。近期有缺陷估计的 Capture- Recapture 模型[5]; 修正的 NHPP 模型[6]; 利用数据挖掘技术提取软件缺陷具有的某些特征[7] 等等。文献[8]将近三十种方法归为静态和动态预测技术两类, 并对现有方法的优劣进行了综述及评价。然而, 众多的缺陷预测模型并不考虑缺陷在软件生命周期中的状态变化, 并且假定某个缺陷的存在并不会带来更多的缺陷。文献[9]通过实验的手段, 验证了缺陷之间确存在关联关系, 并提出一种测试方法用来剔除关联缺陷, 但文中并不考虑缺陷产生的内在机制和演化过程。现有的研究成果表明, 用以刻画缺陷产生及其生长过程的模型还不多见。

文献[10]指出: 软件缺陷的生长过程是内部增长与外部力量和环境条件限制其增长的正负反馈作用的结果。这种激发软件产生缺陷, 促进其增长的因素被称为“失信因子”[11]。失信因子既涉及主观方面, 如: 工作者的技术能力、开发小组的组织管理、程序员的压力等; 也包括客观方面, 如: 项目的规模, 复杂度及难度, 编程语言等。在失信因子的作用下, 软件存在着固有缺陷, 在某些环境、条件的作用下繁殖增长, 又受诸如人、开发过程规范和标准等(以下称为“约束因子”) 的制约。

本文建立在“失信因子”及“约束因子”的共同作用下, 缺陷潜藏、被发现和被修复三种状态变化转移过程的微分方程模型, 用以反映软件缺陷生长的过程机理。同时, 通过分析平衡解的稳定性, 给出“失信因子”的作用边界条件。最后, 用 NASA 中一个项目的数据进行实证分析, 结果表明该模型能较好地反映缺陷的生长过程。

2 问题描述与假设

软件开发本质上是一项集结人类智慧的心智活动, 缺陷在开发过程中产生, 是人类大脑活动的结果。由于软件的自身特点, 以及人的认知局限性, 在

收稿日期: 2010- 01- 26; 修订日期: 2010- 11- 01

基金项目: 国家自然科学基金资助项目(90718036); 江苏省自然科学基金(BK2009232); 南京大学研究生科研创新基金(2009CW06)

作者简介: 占济舟(1982-), 女(汉族), 江西乐安县人, 南京大学工程管理学院在读博士, 研究方向: 复杂系统建模与分析、可信软件项目质量管理.

某个或某些“失信因子”的作用下,软件存在固有缺陷。若没有任何约束,缺陷会随着开发过程的深入不断扩散、繁殖增长,如文献[1]提到:需求阶段中的某个缺陷涉及软件的所有功能,在开发过程中,会随着功能的逐步细化,分布到软件的每个功能中,从而带来缺陷数目的指数增长。

然而,软件开发须遵循一些开发过程规范和标准,以及随着开发人员的经验的积累,开发团队组织管理能力的逐步加强,缺陷不会无限制地增长,这些外部条件在某种程度上控制着缺陷的生长,我们认为这种抑制缺陷生长的现象是由“约束因子”起作用所导致的。

因此,软件开发过程中若不进行软件测试这一工作环节,那么,系统中的缺陷数在“失信因子”和“约束因子”的共同作用下繁殖增长(本文称这种增长过程为缺陷的自增长过程)。一方面,受“失信因子”的激发,系统中潜藏的缺陷呈指数增加;另一方面,“约束因子”对缺陷的增长又起着阻滞作用。缺陷的这种自增长过程与一般情况下的生物种群阻滞增长过程极为相似[12]。

为了使得在软件发布前,系统中可能存在的缺陷尽可能的少,开发团队进行各种测试工作、审查活动(如:单元测试,集成测试,系统测试,走查,同级评审等等)。越早发现软件中潜藏的缺陷,就越有可能快速、容易、经济地修复缺陷。然而,由于缺陷之间存在关联关系,修复缺陷的同时可能给系统带来新的缺陷。因此,软件架构及设计过程中应尽量避免模块之间的耦合,以便于系统的维护。

通过上述的分析,软件缺陷在开发过程中存在三种状态,即:潜藏、被发现、被修复。

单位时间内,若潜藏的缺陷越多,发现的缺陷也越多,随后被修复的缺陷也越多。因此,随着软件开发工作的不断推进,开发人员的认知越来越丰富,开发过程得到改进,缺陷的环境容纳量将逐渐降低。

我们利用软件测试和缺陷修复的信息,对潜藏的缺陷进行预测。为建立软件缺陷的状态转移模型,下面对建模问题及符号进行假设和说明。

(1)忽略软件开发过程中各阶段(需求、设计、编码、测试、维护)中缺陷的差异;

(2)缺陷在软件各构件中的分布大致相同;

(3)每一个缺陷被发现的难易程度大致相同;

(4)忽略缺陷报告的提交时间;

(5) t 时刻,软件潜藏的缺陷数为 $y(t)$;

(6) t 时刻,已发现的缺陷数为 $w(t)$,已修复的缺陷数为 $s(t)$,已发现但未修复的缺陷数为 $x(t)$,$x(t) = w(t) - s(t)$;

(7) t 时刻,缺陷的自然增长率为 $r(t)$,它反映了失信因子的作用强度;约束因子的作用强度为 $b(t)$,它反映了约束因子对缺陷的控制强度;初始环境最大容量为 $r(t)/b(t)$;

(8) t 时刻,单位时间内 $w(t)$ 的变化与 $y(t)$ 成正比,比率系数为 $\beta(t)$;

(9) t 时刻,单位时间内 $s(t)$ 的变化与 $x(t)$ 成正比,比率系数为 $\rho(t)$;

(10) t 时刻,单位时间内,因缺陷修复引入的新缺陷数与修复速率成正比,比率系数为 $\eta(t)$ ($0 \leqslant \eta(t) < 1$)。

3 模型的建立与分析

3.1 模型的建立

由第 2 节的问题分析,容易刻画软件开发中潜藏的缺陷、被发现的缺陷、被修复的缺陷这三种状态的转移过程,如图 1 所示。

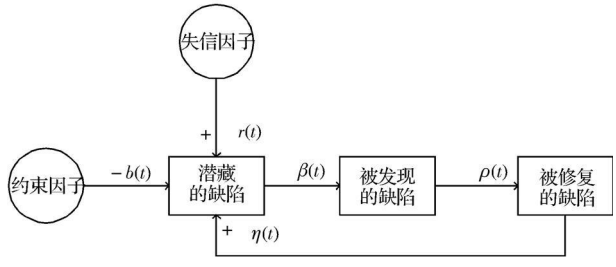

图 1 软件缺陷状态转移过程

首先,不考虑软件测试过程,在“失信因子”和“约束因子”的共同作用下,建立软件缺陷的增长过程(以下称这种增长为“缺陷自增长”)模型。然后,建立软件测试、修复工作执行时的缺陷状态变化的动态模型。

(1)软件缺陷自增长模型

从第 2 节分析可知,软件缺陷自增长过程与生物种群的生长过程相似。在自然界中,种群不可能长期地持续增长。当种群在一个有限的空间中增长时,随着种群密度的上升,对有限空间资源和其他生活必需条件的种内竞争也增加。在环境制约下,种群数量呈现“S”形增长。Logistic 模型在实际问题中,尤其是在生物种群阻滞生长分析研究中得到了广泛应用[12]。

假设一个环境条件下所容许的最大种群数,称为环境容纳量 K;受环境制约的影响,种群的增长密

度上升而逐渐按比例减少, 比例函数为

$$f(N) = (K - N)/K \quad (1)$$

其中, N 为种群数量。

从而构建 N 随时间 t 变化的模型如下:

$$\frac{dN}{dt} = rN[(K - N)/K] \quad (2)$$

其中, r 为每个个体的种群增长率。

积分得:

$$N = \frac{K}{1 + ce^{-rt}} \quad (3)$$

其中, $c = K/N_0 - 1$, N_0 为初始种群数。

式(3)反映了生物种群的增长过程呈"S"形变化, 如图 2 所示。

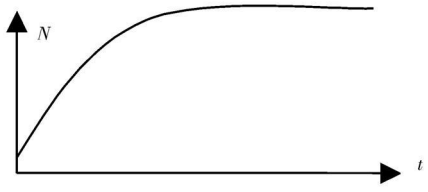

图 2　种群数量增长的 Logistic 模型示意图

类似于种群增长过程, 可建立软件缺陷自增长的 Logistic 模型。在缺陷自增长过程中, 缺陷的环境容纳量由"失信因子"和"约束因子"的共同作

用强度所决定, 本文中设为 $r(t)/b(t)$。缺陷的自然增长率为失信因子的作用强度 $r(t)$。从而, 得到软件缺陷自增长的 Logistic 模型(见(4)式)。

$$\frac{dy}{dt} = r(t)y\left(1 - \frac{y}{r(t)/b(t)}\right) = (r(t) - b(t)y)y \quad (4)$$

(2) 软件缺陷状态转移过程模型

考虑到软件缺陷的特殊性, 缺陷自身不会灭亡, 只有经过软件测试后被修复, 才能将其从系统中移除。下面建立在一般情况下(如图 1), 缺陷状态转移的微分方程模型:

$$\begin{cases} \dfrac{dy}{dt} = (r(t) - b(t)y)y - \dfrac{dw}{dt} + \eta(t)\dfrac{ds}{dt} \\ \dfrac{dw}{dt} = \beta(t)y, \ \dfrac{ds}{dt} = \rho(t)x(t) \\ x(t) = w(t) - s(t) \end{cases} \quad (5)$$

化简得:

$$\begin{cases} \dfrac{dy}{dt} = (r(t) - b(t)y)y - \beta(t)y + \eta(t)\rho(t)x \\ \dfrac{dx}{dt} = \beta(t)y - \rho(t)x \end{cases} \quad (6)$$

3.2　参数的设定

一般地, 记软件测试和修复阶段的几个关键时间段可划分为:

零时刻: 发现缺陷的初始时刻;

t_0: 缺陷修复记录的开始时刻;

t_1: 缺陷被发现的高峰时刻;

t_2: 缺陷被修复的高峰时刻;

t_3: 软件测试工作的结束时刻;

t_4: 缺陷修复工作的结束时刻。

在软件测试初期, 发现缺陷的速率相对较慢, ($\beta(t)$ 相对较小), 随着测试人员对环境的逐步了解, $\beta(t)$ 逐渐增大, 达到顶峰。在开发阶段后期, 系统逐步稳定, $\beta(t)$ 减小, 直到测试工作的停止。缺陷的修复过程与测试过程类似, 为简化计算, 假设在各个时段内, $\beta(t)$ 和 $\rho(t)$ 均为常数, 其变化趋势如图 3 所示。$\beta(t)$ 与 $\rho(t)$ 具体计算过程见第 4 节。

假设在某一时段内, $r(t) = r$, $b(t) = b$, r, b 为常数。随着时间的推进, 开发过程逐步得到完善, $r(t)$ 逐渐变小, $b(t)$ 逐渐变大。随着测试和修复工作接近尾声, 系统的环境基本保持平稳, 此时, $r(t)$ 和 $b(t)$ 保持一定水平的稳定值, 如图 4 所示。

由于缺陷的最大环境容纳量始终大于 1, 所以, $r(t)/b(t) > 1$。一般地, 根据经验, 初始时刻, 取 $r(t)$ 和 $b(t)$ 的初始值为 $0 < b(t) < 1 < r(t)$, $1 \leqslant r(t) \leqslant 3$ (文献[1]中 $r(t)$ 设为 2) 是适应的。

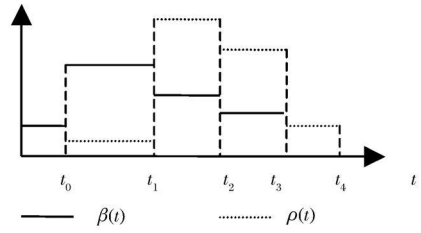

图 3　$\beta(t)$ 与 $\rho(t)$ 的变化趋势

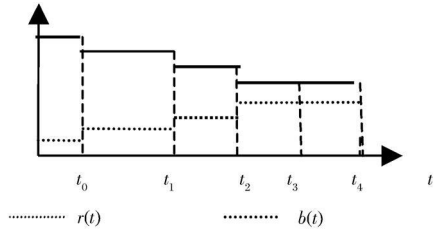

图 4　$r(t)$ 与 $b(t)$ 的变化趋势

通常取 $\eta(t) = \eta$, η 恒为小于 0.05 的常数。由此, (6) 式可简化为:

$$\begin{cases}\dfrac{dy}{dt} = (r_i - b_i y)y - \beta_i y + \eta\rho_i x \\ \dfrac{dx}{dt} = \beta_i y - \rho_i x \quad t_{i-1} \leqslant t \leqslant t_i, i = 1, 2, 3, 4 \\ x(0) = x_0, y(0) = y_0\end{cases} \tag{7}$$

3.3 模型平衡点的稳定性分析

不失一般性, 取(7) 式的一般形式(8) 式进行分析。

$$\begin{cases}\begin{pmatrix}\dfrac{dy}{dt} \\ \dfrac{dx}{dt}\end{pmatrix} = \begin{pmatrix} r - \beta & \eta\rho \\ \beta & -\rho\end{pmatrix}\begin{pmatrix} y \\ x\end{pmatrix} + \begin{pmatrix} -by^2 \\ 0\end{pmatrix} \\ x(0) = x_0, y(0) = y_0\end{cases} \tag{8}$$

显然, 方程(8) 是一平面非线性自治系统, 其解析解难以表达。因此, 研究其平衡点(奇点) 的稳定性及在相平面的轨线走向具有重要意义。令 $dy/dt = 0$, $dx/dt = 0$, 易得方程(8) 的两个平衡点:

$Q_1(0, 0)$, $Q_2 = \left(\dfrac{\beta}{b\rho}(r - \beta + \eta\beta), \dfrac{1}{b}(r - \beta + \eta\beta)\right)$。

方程(8) 所对应的线性系统为:

$$\begin{pmatrix}\dfrac{dy}{dt} \\ \dfrac{dx}{dt}\end{pmatrix} = \begin{pmatrix} r - \beta & \eta\rho \\ \beta & -\rho\end{pmatrix}\begin{pmatrix} y \\ x\end{pmatrix} \tag{9}$$

引理[13]: 设平面非线性系统

$$\begin{pmatrix}\dfrac{dx}{dt} \\ \dfrac{dy}{dt}\end{pmatrix} = \begin{pmatrix} a & b \\ c & d\end{pmatrix}\begin{pmatrix} x \\ y\end{pmatrix} + \begin{pmatrix} X(x, y) \\ Y(x, y)\end{pmatrix},$$

a, b, c, d 为常数, 若 X 与 Y 满足条件:

(1) 在奇点 $O(0, 0)$ 的邻域内有连续的一阶偏导数; (2) $X(x, y) = o(r), Y(x, y) = o(r), r = \sqrt{x^2 + y^2}$, 且 O 是其对应线性系统的焦点、结点或鞍点, 那么 O 也是非线性系统的同类型奇点。

以下对平衡点 Q_1, Q_2 的稳定性进行分析。

对于 $Q_1(0, 0)$:

(9) 式的特征方程为

$$\lambda^2 + (\rho - r + \beta)\lambda + \rho(\beta - \eta\beta - r) = 0 \tag{10}$$

不妨设 $\rho - r + \beta \neq 0$, $r \neq (1 - \eta)\beta$

I) 若 $\beta - \eta\beta - r < 0 \Rightarrow r > (1 - \eta)\beta$ 时, $Q_1(0, 0)$ 是不稳定的鞍点;

II) 若 $\beta - \eta\beta - r > 0 \Rightarrow r < (1 - \eta)\beta$ 时, $Q_1(0, 0)$ 是稳定的结点。

对于 $Q_2 = \left(\dfrac{\beta}{b\rho}(r - \beta + \eta\beta), \dfrac{1}{b}(r - \beta + \eta\beta)\right)$:

令 $u = x - \dfrac{\beta}{b\rho}(r - \beta + \eta\beta)$, $v = y - \dfrac{1}{b}(r - \beta + \eta\beta)$, 则(9) 式化为(11) 式, Q_2 对应于(11) 式的零点。

$$\frac{d}{dt}\begin{pmatrix} v \\ u\end{pmatrix} = \begin{pmatrix} -(r - \beta + 2\eta\beta) & \eta\rho \\ \beta & -\rho\end{pmatrix}\begin{pmatrix} v \\ u\end{pmatrix} \tag{11}$$

(11) 式的特征方程为:

$$\lambda^2 + (\rho + r - \beta + 2\eta\beta)\lambda + \rho(r - \beta + \eta\beta) = 0 \tag{12}$$

不妨设 $\rho + r - \beta + 2\eta\beta \neq 0$, $r \neq (1 - \eta)\beta$

由于 $(\rho + r - \beta + 2\eta\beta)^2 - 4\rho(r - \beta + \eta\beta) = (\rho - r + \beta - \eta\beta)^2 + \eta\beta(2\rho + \eta\beta) > 0$

所以,

I) 若 $\beta - \eta\beta - r < 0 \Rightarrow r > (1 - \eta)\beta$ 时, Q_2 是稳定的结点;

II) 若 $\beta - \eta\beta - r > 0 \Rightarrow r < (1 - \eta)\beta$ 时, Q_2 是不稳定的鞍点。

综上所述, 由引理可得如下结论:

结论: 非线性微分动力系统(8) 若满足 $\rho - r + \beta \neq 0$ 且 $\rho + r - \beta + 2\eta\beta \neq 0$, 有:

(1) 当 $r > (1 - \eta)\beta$ 时, $Q_2 = \left(\dfrac{\beta}{b\rho}(r - \beta + \eta\beta), \dfrac{1}{b}(r - \beta + \eta\beta)\right)$ 为稳定的结点;

(2) 当 $r < (1 - \eta)\beta$ 时, $Q_1 = (0, 0)$ 为稳定的结点。

可以看出: 假若平均修复 100 个缺陷会引进一个缺陷(即 $\eta = 0.01$), 则:

(1) 当 $r < 0.99\beta \approx \beta$ 时, 也就是说, 失信因子的激发强度不如测试人员发现缺陷的能力, 那么 $t \to \infty$ 时, 所有的缺陷都会被发现、被修复。这意味着, 一个组织规范合理, 专业技术高, 拥有一个优秀测试团体的开发组织, 能够开发出较好质量的软件。

(2) 当 $r > 0.99\beta \approx \beta$ 时, 即失信因子的作用强度高于测试人员的发现能力, 那么, 当 $t \to \infty$ 时, 软件的潜藏缺陷趋于 $(r - \beta)/b$。由问题的假定, 可知 $(r - \beta)/b$ 表示潜藏缺陷最终的环境容纳量。要使得 $(r - \beta)/b$ 较小, 需要增大 b (即加强规范约束), 这取决于开发商的能力成熟度。也就是, 若开发小组中, 测试人员发现缺陷的能力不能提高, 那么, 在软件开发的各个阶段中, 工作人员都必须严格按照规范章程, 尽量提供一个良好的设计平台, 增强开发人员的责任感, 使得软件在开发过程中的缺陷得以控制。

3.4 失信因子的边界条件分析

对于上述模型, 假定软件开发过程中缺陷修复速率与缺陷发现速率相当, 即针对每个发现的缺陷均作修复。在 $b(t)$ 为常数的假设下, 失信因子的作用强度表征了软件开发过程中外界环境的优劣, 从另一方面也反映了开发团队、开发人员在软件质量控制上的努力程度。优化软件开发平台, 提高软件开发人员的工作能力, 将导致软件开发过程管理成本的增加, 软件团队、开发人员对质量控制的努力程度的提高意味着软件团队与个人时间、精力投入的增加, 为此需要研究失信因子作用强度至多达到何种水平, 才可保证软件缺陷数量在软件发布时达到预期目标, 即需要对模型中的失信因子进行边界条件分析。

令 $\frac{dx}{dt} = \beta y - \rho x = 0$, 代入 $\frac{dy}{dt} = ry - by^2 - \beta y + \rho\eta x$ 中, 得:

$$\frac{dy}{dt} = ry - by^2 - \beta y + \eta\beta y = (r - (1-\eta)\beta)\left(1 - \frac{1}{\frac{r-(1-\eta)\beta}{b}}y\right)y \quad (13)$$

令 $dy/dt = 0$, 得到缺陷的稳定量为 $(r - (1-\eta)\beta)/b$。假设软件潜藏的缺陷目标为 y_c, 约束因子的限制能力稳定为常数 b, 则:

$$\frac{r-(1-\eta)\beta}{b} \leqslant y_c \Rightarrow r \leqslant by_c + (1-\eta)\beta \quad (14)$$

由(13) 式可知: 当 η 充分小时(通常也称为缺陷的修复是完美的), r 的上界为 $by_c + (1-\eta)\beta$。

y_c 越小, 要求失信因子的作用能力也越弱(即约束因子的作用相对较强, 开发规范较完善); 若 β 越小, 说明测试人员发现缺陷的能力较弱, 只有当失信因子的作用强度也越弱时, 才能达到预期目标, 这与实际情况相符。

为验证本文所提出的模型, 采用美国宇航局(NASA) 中一个环球卫星的飞行软件的缺陷数据[14] 进行实例分析。

4　实证分析

PC1 为 NASA 的一个环球卫星飞行软件。代码使用 C 语言便携, 代码规模约为 40KLOC, 共有 1107 个模块, 平均循环复杂度为 5. 52, 缺陷数据见表 1。

表 1　PC1 的缺陷数据

	时间 (MM-YY)	发现缺陷	累计发现	缺陷修复	累计修复	未修复		时间 (MM-YY)	发现缺陷	累计发现	缺陷修复	累计修复	未修复
0	10- 98	1	1				23	09- 00	20	283	59	180	103
1	11- 98		1				24	10- 00	15	298		180	118
2	12- 98	2	3				25	11- 00	18	316	39	219	97
3	01- 99	3	6				26	12- 00	8	324	15	234	90
4	02- 99	7	13				27	01- 01	9	333	24	258	75
5	03- 99		13				28	02- 01	11	342	2	260	82
6	04- 99		13				29	03- 01	22	364	2	262	102
7	05- 99	1	14				30	04- 01	4	368		262	106
8	06- 99		14				31	05- 01	38	406	17	279	127
9	07- 99		14				32	06- 01	9	415	52	331	84
10	08- 99	2	16				33	07- 01	11	426		331	95
11	09- 99		16				34	08- 01	29	455	21	352	103
12	10- 99	1	17				35	09- 01	12	467	61	413	54
13	11- 99	18	35				36	10- 01	3	470	15	428	42
14	12- 99	10	45	11	11	34	37	11- 01	5	475		428	47
15	01- 00	12	57		11	46	38	12- 01	9	484		428	56
16	02- 00	11	68		11	57	39	01- 02	24	508		428	80
17	03- 00	38	106		11	95	40	02- 02	16	524	8	436	88
18	04- 00	18	124		11	113	41	03- 02	13	537	54	490	47
19	05- 00	37	150	23	34	116	42	04- 02	3	540	4	494	46
20	06- 00	33	183	62	96	87	43	05- 02	13	553		494	59
21	07- 00	32	215		96	119	44	06- 02	5	558	25	519	39
22	08- 00	48	263	25	121	142	45	07- 02	3	561		519	42

注: 空白处表示“0”。

利用微分动力系统模型预测软件潜在的缺陷数:

$$\begin{cases} \dfrac{dy}{dt} = (r_i - b_i y)y - \beta_i y + \eta\rho_i x \\ \dfrac{dx}{dt} = \beta_i y - \rho_i x, \quad t_{i-1} \leqslant t \leqslant t_i, i = 1, 2, 3, 4 \\ x(0) = x_0, y(0) = y_0 \end{cases}$$

其中, 设定 $\eta = 0.01$。其它各参数的确定如下:

记 $\Delta t_i = t_i - t_{i-1}$, 从表 1 可以看出, 初始时刻设为 1998 年 10 月, $t_0 = 14$, $t_1 = 22$, $t_2 = 35$, $t_3 = 44$, $t_4 = 45$。

$\beta_i = \dfrac{N_i}{(1000 - N_i)\Delta t_i}$, 其中 N_i 为 Δt_i 时段内的缺陷累积发现数。

$\rho_i = \dfrac{M_i}{S_i \Delta t_i}$, 其中 M_i 为 Δt_i 时段内, 已修复缺陷的总数; S_i 为 t_i 时刻, 已发现但仍未修复的缺陷总数。计算得 β_i, ρ_i 的值如下:

$$\beta = \begin{cases} 0.002591, & 0 \leqslant t \leqslant 14; \\ 0.034374, & 14 < t \leqslant 22; \\ 0.029441, & 22 < t \leqslant 35; \\ 0.021412, & 35 < t \leqslant 45 \end{cases}$$

$$\rho = \begin{cases} 0, & 0 \leqslant t \leqslant 14; \\ 0.094679, & 14 < t \leqslant 22; \\ 0.415954, & 22 < t \leqslant 35; \\ 0.252381, & 35 < t \leqslant 44 \end{cases}$$

假设初始值 $r_0 = 2$, $b_0 = 0.002$, 各阶段的 r_i, b_i 的取值不妨假定为:

$$r = \begin{cases} 2.0, & 0 \leqslant t \leqslant 14; \\ 1.6, & 14 < t \leqslant 22; \\ 1.2, & 22 < t \leqslant 35; \\ 0.8, & 35 < t < \infty \end{cases},$$

$$b = \begin{cases} 0.002, & 0 \leqslant t \leqslant 14; \\ 0.006, & 14 < t \leqslant 22; \\ 0.010, & 22 < t \leqslant 35; \\ 0.050, & 35 < t < \infty \end{cases}$$

由于模型的解析解难以得到, 本文利用 MAPLE 数学软件得到各个时段的 $x(t)$, $y(t)$ 在其相平面的变化趋势(其中, 图 6—图 8 描绘了 $x(t)$, $y(t)$ 的相平面, 其中每一条轨线表示 $x(t)$ 取某一值时, $y(t)$ 的变化趋势, 轨线箭头指向了 $y(t)$ 的平稳值), 如图 5—图 8 所示。

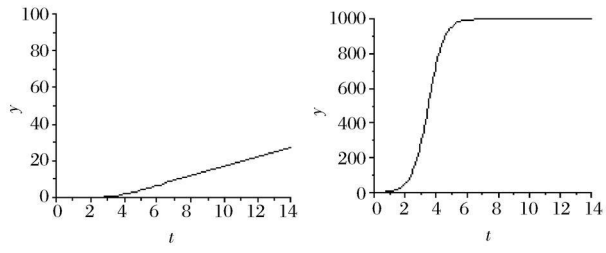

图 5 $0 < t \leqslant 14$, $x(t), y(t)$ 的趋势图

表 1 中, $t = 14$ 时, 系统中有 34 个缺陷未被修复。这与图 5 中 $x(14)$ 的值基本接近。当缺陷测试和修复均处于初始阶段时, 系统中潜藏的缺陷远远大于系统中已知存在的缺陷, 图 5 中 $y(t)$ 曲线的走向正反映了这一客观事实。

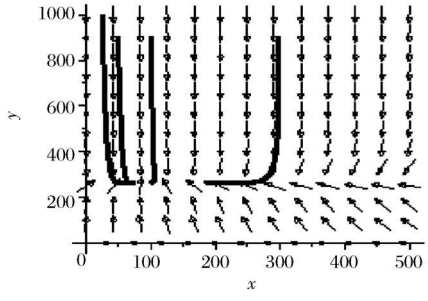

图 6 $14 < t \leqslant 22$, 相平面 $x(t), y(t)$ 的轨线图

从表 1 中可知, PC1 项目中累计发现了 561 个缺陷, 当 $t = 22$ 时, 共发现了 263 个缺陷。这就是说, 系统中潜藏未知的缺陷至少有 $561 - 263 = 298$ 个缺陷。此时, 系统中仍未修复的缺陷 $x(22) = 142$。从图 6 中 $y(t)$ 轨线的走向可知, $y(22)$ 基本稳定在 300, 接近于 298。

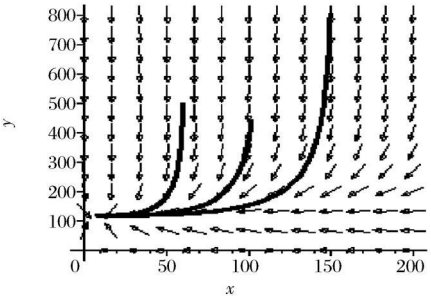

图 7 $22 < t \leqslant 35$, 相平面 $x(t), y(t)$ 的轨线图

表 1 中可以看出, 当 $t = 35$ 时, 共发现了 467 个缺陷。这就是说, 系统中潜藏未知的缺陷至少有 $561 - 467 = 94$ 个缺陷。此时, 系统中仍未修复的缺陷 $x(35) = 54$。从图 7 中 $y(t)$ 轨线的走向可知, $y(35)$ 基本稳定在 100, 接近于 94。

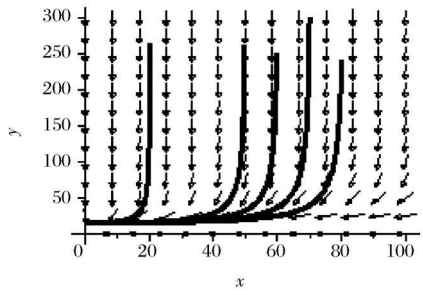

图 8　$t>35$, 相平面 $x(t), y(t)$ 的轨线图

当 $t>35$ 时, 软件测试和修复工作都接近尾声, 系统的环境基本保持平稳。从表 1 中可以看到, 系统中仍有 $x(45)=42$ 个缺陷待修复。通过本文 3.3 节的分析, 我们可对今后系统潜藏的缺陷进行以下预测:

因为 $r=0.8$, $(1-\eta)\beta=0.99\times0.021412=0.021198$, $r>>(1-\eta)\beta$。因此, $Q^2=(\frac{\beta}{b\rho}(r-\beta+\eta\beta), \frac{1}{b}(r-\beta+\eta\beta))$ 为平衡点, 即 $x(t)\rightarrow 1.36$, $y(t)\rightarrow 16$。也就是说, 当时间充分长时, 发现的缺陷几乎都将被修复, 软件系统中可能潜藏约 16 个缺陷, 如图 8 中 y(t) 的变化趋势所示。

5　结语

软件缺陷预测一直是软件质量工程关注的热门话题, 然而, 目前的预测技术并未考虑缺陷在软件生命周期中的状态变化。本文从更深层次的角度对缺陷进行分析。首先, 基于软件缺陷产生及生长过程的机理, 建立了一个在失信因子和约束因子共同作用下, 关于缺陷潜藏、被发现以及被修复三种状态动态转移的微分方程模型; 然后, 对模型的平衡点的稳定性进行了分析, 给出在一定的预期目标下, 失信因子的边界条件; 最后, 利用 NASA 的实际数据对模型进行仿真, 表明模型的合理性。本文的研究工作提出了一种利用缺陷生长过程中的不同状态对潜藏缺陷进行预测的新思路, 进一步丰富了现有的软件缺陷预测理论和软件质量评估理论, 为软件项目的发布提供了一个实际参考, 同时, 为在软件开发过程中, 失信因子的控制研究垫定了理论基础。

由于缺陷在生长过程中的动态预测研究尚处于初步阶段, 一些研究问题还有待今后进一步的探讨。比如: 如何正确地分析软件测试、缺陷修复过程不同时段的划分对缺陷预测模型的影响; 如何正确地刻画失信因子对缺陷的作用机理, 在软件开发实践中, 应对失信因子施予什么样的控制策略; 以及如何建立在不同失信因子相互作用下, 缺陷的状态转移模型等等。

参考文献:

[1] 聂剑平, 韩柯, 陈光, 曹旭. 软件缺陷增长过程的混沌分析[J]. 计算机工程与应用, 2008, 44(11): 97-100

[2] Fenton, N. E.. Software Metrics: Successes, failures and new directions [J]. Journal of Systems and Software, 1999, (47): 149-157.

[3] Fenton, N. E., Martain, N., William, M., Peter, H., Lukrad, R., Paul, K.. Predicting software defects in varying development lifecycles using Bayesian nets [J]. Information and Software Technology, 2007, 49(1): 32-43

[4] Goel, A. L., Okumoto, K.. A time-dependent error detection rate model for software reliability and other performance measures [J]. IEEE Transaction on Reliability, 1979, R-28(3): 206-211.

[5] Emam, E. K., Laitenberger, O.. Evaluating capture-recapture models with two inspectors [J]. IEEE Transaction on Software Engineering, 2001, 27(9): 851-864

[6] Bai, C. G., Cai, K. Y., Hu, Q. P., Ng, S. H.. On the trend of remaining software defect estimation [J]. IEEE Transaction on Systems, Man, and Cybernetics-PART A: Systems and Humans, 2008, 38(5): 1129-1142

[7] Song, Q. B., Shepperd, M., Cartwright, M., Mair, C.. Software defect association mining and defect correction effort prediction [J]. IEEE Transaction on Software Engineering, 2006, 32(2): 69-82

[8] 王青, 伍书剑, 李明树. 软件缺陷预测技术[J]. 软件学报, 2008, 19(7): 1565-1580.

[9] 景涛, 江昌海, 胡德斌, 白成刚, 蔡开元. 软件关联缺陷的一种检测方法[J]. 软件学报, 2005, 16(1): 17-28

[10] 张凯, 熊前兴. 软件缺陷混沌分形生长与机理分析[J]. 武汉理工大学学报, 2004, 26(1): 80-82

[11] Zhan, J. Z., Zhou, X. Z., Zhao, J. B.. Analysis of the original cause of software distrust [C]. The 2nd International Conference of SEDM, Chengdu, China, June, 2010, 240-245

[12] 马知恩. 种群生态学的数学建模与研究[M]. 合肥: 安徽教育出版社, 1996

[13] 马知恩, 周义仓. 常微分方程稳定性与稳定性方法[M]. 北京: 科学出版社, 2001.

[14] http://nssdc.gsfc.nasa.gov/.

A Model for Predicting Software Defect Based on Distrustable Factor

ZHAN Ji-zhou[1], ZHOU Xian-zhong[1], ZHAO Jia-bao[1], WANG Jian-feng[2]
(1. School of Management and Engineering, Nanjing University, Nanjing 210093, China;
2. Department of mathematics, Nanjing University, Nanjing 210093, China)

Abstract: Software defect is the key measure of software reliability, and predicting defect is also one of the core topics of software quality evaluation. Based on analysis of mechanism of software defect growth, the states of defect are divided into hidden, be detected and repaired. Considering the combined effect of distrustable factors and constrained factors, the differential equation model for defect states transferring is proposed in this paper. The stability of equilibrium point and boundary condition of distrustable factors are also analyzed. Meanwhile, a real project of NASA is analyzed, the hidden defects are estimated by the record of detected and repaired defect, and the results are exemplified the rationality of the model.

Key words: defect; distrustable factor; differential equation; prediction