基于过程模型随机仿真的 TDD 模块选取建模方法研究 *

苏 峰[1,2], 翟 健[1,2], 杨秋松[1]
(1. 中国科学院 软件研究所 互联网技术实验室, 北京 100190; 2. 中国科学院 研究生院, 北京 100049)

摘 要: 为了帮助项目经理合理选择 TDD 实施模块,基于进程代数方法对测试驱动软件开发过程和非测试驱动软件开发过程建立过程仿真模型。通过用例度量软件模块的复杂性来获取随机变量参数对模型调参,并采用该模型得到仿真结果。提出 TDD 模块选取算法来分析仿真结果并得出最佳 TDD 实施策略,最终为项目经理提供合理的决策。

关键词: 软件过程; 随机进程代数; 测试驱动开发; 仿真; 决策支持
中图分类号: TP311.52 **文献标志码**: A **文章编号**: 1001-3695(2010)08-2948-05
doi:10.3969/j.issn.1001-3695.2010.08.037

Modeling and simulation of TDD components selection using stochastic process algebra

SU Feng[1,2], ZHAI Jian[1,2], YANG Qiu-song[1]
(1. *Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China*; 2. *Graduate School, Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract: In order to help PM selecting the best TDD implement components, this paper developed a simulation model of TDD based on stochastic process algebra. The model was tuned using stochastic parameters derived from complexity metrics of using cases. This paper used this model to getting simulation results. The simulation results could be analyzed to get an optimal TDD development strategy. Ultimately the strategy could provide decision support for project managers.
Key words: software process; stochastic process algebra; test driven develop; simulation; decision support

0 引言

随着信息技术的发展,在软件开发和维护的过程中遇到了一系列严重的问题,如怎样满足日益增长的需求,如何维护不断膨胀的已有软件,这些问题被称为软件危机。为了解决这些问题,学术界和工业界提出了一系列的软件工程方法,其目的主要是提高软件产品质量,保障项目进度,降低项目成本,减少维护费用。基于软件产品质量和软件开发过程质量直接相关的广泛共识,一个重要的研究方向是对软件开发过程本身的研究和改进,该研究领域称为软件过程[1]。通过软件过程研究,能为评估、支持和改进软件开发活动提供方法和技术。

测试驱动开发[2,3](test driven development, TDD)就是近些年来提出的敏捷开发中一种非常流行的方法。它的重点是根据将要开发的程序要求先写好测试用例,并且在开发过程中不断地运行测试来获得所开发的代码与所要求的结果之间的差距。TDD 有别于传统的先编码后测试的开发过程,在开发前先编写测试用例,然后通过自动化测试工具运行测试用例。在未有任何实际功能代码前的测试用例是通不过的,为了使该测试用例能够正确执行,就要对代码进行编写、修改,直到符合测试用例的要求,然后再按照这种方式进行新的功能模块的开发。测试驱动开发(TDD)的以下观点被广泛认可[4]:

a)快速反馈。测试先行可以令开发人员检查新增功能是否满足需求,是否与原有功能冲突。

b)任务驱动。测试驱动开发活动,鼓励开发者分解任务,把复杂任务分解为更加可控、可以实现的简单任务;帮助开发者关注需求要点,使开发过程更加稳定、可度量。

c)质量保证。测试先行可以保证在某种层次的产品质量,先行的功能测试粒度小,更容易反复修改执行。

d)较低层次设计。测试先行提供了进行较低层次设计决策的环境,为代码重构提供了基础。

测试驱动开发可以提高软件产品质量的观点是被广泛认同的,但是其提倡者提出的能够提高开发人员的开发效率方面的观点还是受到质疑。测试驱动开发不可避免地增加代码规模,延长项目配置时间(因为需要更多工具、为测试设备提供支持等),需要投入足够的时间编写测试,需要有合适的工具、有经验的人等。由于以上原因,在项目中使用 TDD 既是对开

收稿日期: 2010-01-27; **修回日期**: 2010-03-03 **基金项目**: 国家自然科学基金资助项目(90718042,60903051);国家"863"计划资助项目(2007AA010303);国家"973"重点基础研究发展计划基金资助项目(2007CB310802);中国科学院知识创新工程领域前沿资助项目(ISCAS09-DR09)

作者简介:苏峰(1983-),男,河北保定人,硕士,主要研究方向为软件过程方法与技术(sufeng@itechs.iscas.ac.cn);翟健(1981-),男,博士,主要研究方向为软件过程方法与技术;杨秋松(1977-),男,助理研究员,博士,主要研究方向为软件过程方法与技术.

发人员自信心的挑战也是对开发人员素质的挑战,在实际的软件项目管理中,所面临的一个重要风险就是项目延期风险。尤其在项目后期中,为了应对延期风险,项目经理往往依据经验或者直觉采取增加工作强度、增加人手、缩减不必要的功能等措施。这种随意决策,往往导致代码质量下降、开发人员过度疲劳、项目组出现不稳定因素,甚至导致项目失败。

文献[5]研究表明,80% 的缺陷往往发生在 20% 的模块中,而模块越复杂,其缺陷产生的概率越高。在实际项目中由于进度压力,严格的测试驱动开发往往不能自始至终贯彻执行。由于零缺陷只是一种理想状态,没有必要为了达到零缺陷而对每个开发模块进行高覆盖率的 TDD。如何合理进行 TDD 开发对于每个项目负责人来说是一个无法回避的问题。

软件过程仿真(software process simulation)无论在学术界还是工业界越来越为软件工程界所重视[6]。它可以帮助软件开发人员更好地理解和评估新的软件工程方法在实际项目中的效力,并预测在开发过程中可能遇到的问题,从而避免其在实际项目中发生。近些年来,软件过程仿真开始用于解决软件开发策略问题,并且为项目经理提供过程改进支持。针对 TDD 方法的评价和仿真研究[2~4]取得了很多成果,这些研究为在实际项目中更加深入地了解和应用 TDD 打下基础。建立适用于实际项目的仿真模型,对于更好地研究、使用和评价测试驱动开发方法十分重要。

为了降低项目风险,本文通过随机仿真方法研究比较每个软件模块 TDD 开发过程和非 TDD 开发过程,从而选取 TDD 实施模块。

1　整体框架

图 1 展示了本文工作的整体框架。由于软件开发时间和缺陷密度与所开发的模块复杂性紧密相关,而 UML 中的用例图和协作图可以用于估计模块的复杂度。本文将通过 UML 中的用例图和协作图来度量软件开发模块的复杂度,从而得到随机进程代数仿真中的随机参数;通过 TDD 模块选取算法,从仿真数据中选择最合理的 TDD 实施方案,即哪些模块采取 TDD 方式开发,哪些模块采取传统方式开发可以获得整体上的最大收益。本文后面的内容将对模块复杂性计算、随机进程代数仿真模型、TDD 模块选取算法作详细的描述。

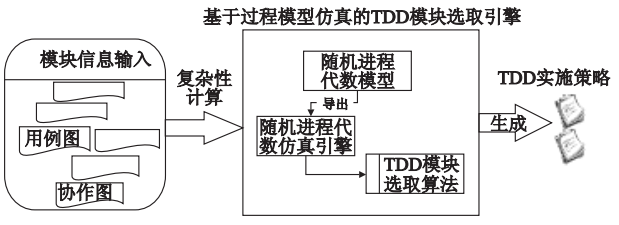


图1　整体框架概要

2　模块复杂度度量

软件系统的结构刻画和度量,将软件系统抽象成有向图成为软件工程领域常用的系统分析方法,结构信息的度量和刻画是软件模块复杂度的重要组成部分。软件模块的结构复杂度可以看成外部复杂度。现在软件系统大部分采用层次化的、模块化的开发方法;各个模块的复杂度,不仅与其自身的内部逻辑复杂度相关,而且和模块外部与之相连的模块亦相关。在软件开发的需求分析和系统设计阶段,人们一般依据用例设计各个功能模块,每个模块不是孤立的,如何度量模块的外部复杂性,从三个方面考虑,即连接度、抽象度、边权重[7]。整个系统复杂度不仅取决其自身的逻辑复杂度,而且还与它进行协作的元素复杂度相关。

2.1　模块结构复杂度度量

模块结构复杂度主要考虑模块的外部特性,即哪些模块与之关联,模块的粒度、模块的连接权重。本文主要从两个方面考虑模块的结构复杂度。

2.1.1　模块连接度

模块的连接度是其结构复杂度的重要参数,在软件系统结构有向图中,本文考虑的连接度是模块的出度和入度之和。本文主要利用 UML 里的协作图来计算模块的连接度。一个节点的连接度表示一个模块与其他模块的紧密程度。本文采用模块的结构熵[8]来替换节点的连接度

$$q_i = d_G(v_i)$$

$$E_{v_i} = -(d_G(v_i)/2L)\ln(d_G(v_i)/2L) = -\frac{d_G(v_i)\ln d_G(v_i)}{2L} + \ln 2L$$

其中:$d_G(v_i)$表示在协作图中节点 v_i 的边的数目;L 为协作图中的总边数;$d_G^{+}(v_i)$表示节点 v_i 的入度;$d_G^{-}(v_i)$表示出度。对于有向图来说:

$$\sum_{i=1}^{n} d_G^{+}(v_i) = \sum_{i=1}^{n} d_G^{-}(v_i) = L$$

2.1.2　边的权重

模块的复杂度不光与它的连接度有关,还同与之相连的模块的复杂度相关,所以这里有复杂度传递的问题,为了简化模型,简单考虑复杂度的传递,即用边的权重代替复杂度传递。

边的权重表示它对应的模块间关系的重要性,记为 $w_G(e)$,计算方法 $w_G(e):f(w_i,\theta)\rightarrow R$,其中:$\theta$ 为阈值;f 作为映射函数,本文中为简单的线性函数,$\sum w_G(e)=1$。

在考虑边的权重后,变为加权有向图,增加了计算的难度。

最终模块的外部结构复杂度可计算如下:

$$R_M = -\frac{1}{2}\sum_{i\in M}\frac{w_i}{2}\ln\frac{1}{2}\sum_{i\in M} w_i$$

2.2　模块内部逻辑复杂度度量

代码逻辑复杂度主要考虑其内部实现逻辑,本文对逻辑复杂度的度量主要利用 UML 中的用例图进行估计。用例建模是一项业界广泛采用的技术,被用于描述和捕捉软件系统的功能需求。用例不仅是一种获取和描述需求的方法,而且是一种测试系统的方法,可以由用例图来估计模块的逻辑复杂度。

Use case(用例)是 UML 中一个非常重要的概念,在使用 UML 的整个软件开发中,use case 处于核心地位。在 UML 的文档中,use case 的定义是:在不展现一个系统或者子系统内部结构的情况下,对系统或者子系统某个连贯的功能单元的定义和描述。Use case 可以用多种方法来描述,可以用自然语言、形式化语言或者是图例。最常用的是用例图,如图 1 所示。用例图是由参与者(actors)、用例(use case)和事件流,以及关系组成。

1)参与者　直接与系统相互作用的系统、子系统或类的外部实体的抽象。它是用户所扮演的参与者,是系统的用户,每个系统定义了一个参与者集合。

2)用例　对一组动作序列的描述,系统通过执行这一组

动作序列为参与者产生一个可观察的结果,是系统具有的一种行为模式,说明了一个参与者与系统执行的一个相关的事务序列。

3)事件流　它是用例完成需求行为的事件描述。事件流的目的是建立用例中逻辑流程的文档,详细描述系统用户的工作和系统本身的工作,既包括正常状态下系统完成需求行为的事件,也包括在其他状态下不能完成需求行为的事件(图2)。

通过分析用例模型中的事件流估算软件的逻辑复杂度。具体计算步骤如下:

a)计算模块中用例个数;b)计算用例中的事务数;c)考虑技术因子和环境因子进行修正(表1)。

事件流描述
1 顾客点击按钮选择"下定单"
2 系统显示输入界面
3 顾客输入想要订购的产品号码
4 系统显示该号码产品的描述和价格
5 顾客输入产品的数量
……

图2　用例事件流描述

表1　修正因子

修正因子	说明
T_1	分布式系统
T_2	组件相应或吞吐能力
T_3	最终用户效率
T_4	复杂内部处理
T_5	代码必须可以重用
T_6	易于修改、安装
T_7	特殊安全性

逻辑复杂度公式为

$$L_M = \sum_{i=0}^{N_m}((1+\sum T_i)\times\sum_{j=0}^{UC_i} UC[i][j])$$

其中:$UC[i][j]$表示第 i 个用例中第 j 个事件的复杂度,默认值为1,其总和在默认情况下表示用例中的事件个数;T_i 表示表1中的修正因子,用于对该模块中的用例进行修正。

模块总体复杂度以内部逻辑复杂度为主,最终复杂度度量公式为

$$C_{PM} = \theta \times R_M + (1-\theta)\times L_M$$

本文中的度量方法不用考虑每个用例的层次及继承度,方便用于模拟。最后对所有模块的复杂度进行归一化处理,可以得到该模块在随机进程代数仿真中的随机参数 γ 值:

$$C_P \propto \gamma$$

上面表示模块归一化后的复杂度与随机进程代数的随机参数正相关。

3　随机进程代数仿真

3.1　随机进程代数仿真模型描述

3.1.1　随机进程代数(stochastic process algebra,SPA)

随机进程代数继承了进程代数对模型的代数形式的描述方法,为系统模型定义了一套完整的语法与语义。其语法中的基本元素是组件和活动,组件由大写字母表示,活动由触发活动动作名称和活动速率两个参数表示,目前有几种典型的随机进程代数模型描述语言,如 TIPP(timed process and performance analysis)[9]、PEPA (performance evaluation process algebra)[10]等。这几种建模方法的主要区别是存在多个活动进行同步时,最终实施活动的速率计算方法不同于语法语义中一些操作定义的差别。这些建模方法已经有工具实现,如 PEPA Eclipse Plug-in[11]实现了 PEPA。

多元随机 π 演算最早由 Herzog[12]在1990年提出,是对经典进程代数如 CCS[13]、CSP[14],或者是 π 演算的一种扩展。它在传统的进程代数基础上添加了时间的随机信息,主要用于系统的功能设计和性能分析的建模。多元随机 π 演算广泛应用于化学和生物反应系统的建模和仿真[15]。

定义1　随机多元 π 演算的语法由下面的 BNF 等式给出。

$$P := M \mid P \mid P' \mid (\nu z)P \mid !P$$

$$M := 0 \mid \pi.P \mid M + M'$$

$$\pi := \bar{x}(y) \mid x(z) \mid (\tau, r) \mid [x=y]\pi$$

其中:0 是一个动作,表示过程 P 什么都不做;$\pi.P$ 中前缀表示输入、输出,或者隐藏动作,或者匹配动作后导致 P 发生;$M+M'$表示选择关系;$P|P'$表示过程 P 和 Q 并行,独立,但是 P 和 Q 可以互相通信;$\bar{x}(y)$和 $x(z)$表示 P 通过通道 x 发送 y;$x(z).Q$ 表示 Q 通过通道 x 接受 y,转为 $Q\{y/z\}$;不可见前缀$(\tau,r).P$ 表示 P 不可见地进入 P,$(\tau,r).P$ 中,τ 表示一个内部动作,对于外部观察者不可见,$r\in(0,+\infty)$是负指数分布中参数,该部分表示活动的持续时间,也可理解为速率;$+\infty$ 由符号 T 表示,意味着活动瞬时完成;0 表示活动需要一个足够的时间完成,或者表示永远不可完成;匹配前缀$[x=y].P$ 表示只有在 x 和 y 的名字相同时才引发 P,否则如动作0。

3.1.2　s-TRISO/ML[16]带有随机信息建模语言及其转换规则

s-TRISO/ML 建模描述语言用于描述软件过程的随机性质和仿真过程模型性能。s-TRISO/ML 包括两个部分:随机进程代数语言,以随机多元 Pi 演算为基础;转换规则,用于计算活动速率和操作定义的状态间的转换。

s-TRISO/ML 可以把仿真的过程模型映射为多元随机 π 演算[16,17],可方便建模人员在不必熟悉复杂的多元随机 π 演算形式化文法的前提下,建立基于多元随机 π 演算的模型并生成复杂的多元随机 π 演算表达式,为后续对模型验证和仿真提供支持。在 s-TRISO/ML 中,一个软件过程由一系列的相互交换的实体并发组成,而每个实体的描述包括了实体的内部行为以及与周围其他实体的交互。转换规则包括人员转换规则、行为节点转换规则、顺序行为节点转换规则、并发节点转换规则、选择行为节点转换规则、终端行为节点转换规则、实体节点转换规则、辅助过程转换规则。本文只是针对 TDD 方法进行仿真来获取 TDD 开发策略,因此忽略整个团队人员变化,不考虑人员转换规则。

3.2　TDD 随机进程代数仿真模型

对于每个模块(组件)C_i,本文在建模时考虑其约束 R_i 为一个约束向量 (T_i, Q_i, A_i)。其中:T_i 表示进度约束,是在项目开发中对模块 C_i 限制的开发时间;Q_i 是模块的质量约束,主要由模块的缺陷数量来度量,与模块的复杂度正相关,与测试强度负相关;A_i 表示一个模块的活动集合(其主要考虑四种活动:A 编程活动(CodeAct),此项工作完成的产品提交到版本库中的源代码;B 模块功能增添,重构并迭代活动(RefactorAct);C 测试工作(TestAct),表示测试某些代码并提交缺陷报告;D 调试工作(DebugAct),根据缺陷报告解决代码中发现的缺陷)。

对于每个模块通过前文的方法计算出其复杂性的一个度量值,这个值反映模块采用 TDD 和非 TDD 对于模块开发时间和模块缺陷数目的影响。TDD 对模块开发进度的影响作为一个满足特定分布(本模型中为指数分布)的随机变量 τ,缺陷数目为另一个随机变量 δ,模型加入 TDD 后,可有如下好处:

a)编码的平均时间增加,即使速率变慢。

b)引入缺陷的数目减少。

c)缺陷定位时间减少,修复一个缺陷的调试时间减少,并且减少潜在缺陷数目。

对于一个模块 A,采用 TDD 或者 non-TDD 开发过程如图 3 所示,可以分解为上面提到的四个子活动,可以建立其 s-TRISO/ML 仿真模型,→节点表示顺序关系,v 表示虚节点,+ 节点表示选择关系,Dum 是哑元节点,表示活动瞬时发生(如图 4、5 所示)。

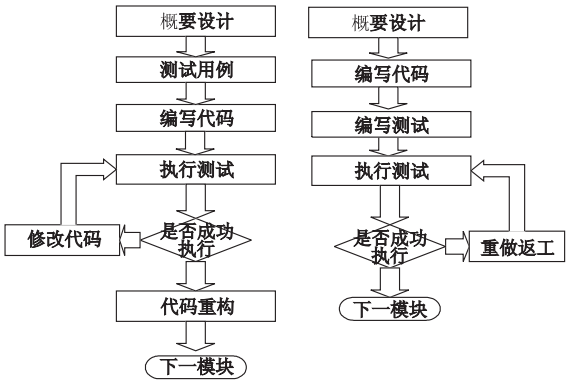
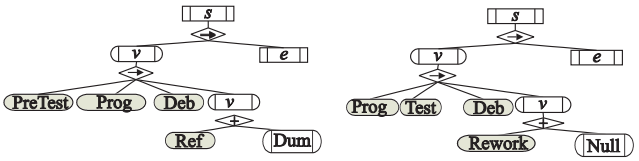


图3　TDD开发流程与传统开发流程



s-TRISO/ML 模型可以转换为随机进程代数(SPA)如下:

对于 TDD 开发:

$P_{\mathrm{TDD}} := (\mathrm{preTest},\delta).(\mathrm{program},p_1\lambda).(\mathrm{debug},p_2\rho).P_{\mathrm{debug}}$

$P_{\mathrm{debug}} := (\mathrm{ref},\zeta).P'_{\mathrm{TDD}} + (\mathrm{pass},\perp).P_{\mathrm{end}}$

$B := (\mathrm{debug},f_T).B' + (\mathrm{end},\perp).B_{\mathrm{end}}$

$P_{\mathrm{TDD}}\langle \mathrm{debug}\rangle B$

对于 non-TDD 开发:

$P_N := (\mathrm{program},p'_1\lambda).(\mathrm{Test},\delta').P_m$

$P_m := (\mathrm{debug},p'_2\rho).P_{\mathrm{debug}}$

$P_{\mathrm{debug}} := (\mathrm{rework},\perp).P'_N + (\mathrm{pass},\perp).P_{\mathrm{end}}$

$B := (\mathrm{debug},f_N).B'_{\mathrm{fix}} + (\mathrm{end},\perp).B_{\mathrm{end}}$

$P_N\langle \mathrm{debug}\rangle B$

其中:P_{TDD} 和 P_N 表示选择 TDD 方式还是传统方式开发过程,这两种开发方式的仿真参数不同,TDD 方式开发按图 4 执行;preTest 是先行测试活动标志;$p_1\lambda$ 是 TDD 开发和模块复杂性相关的随机变量的参数,该随机变量服从负指数分布,其倒数与活动的期望持续时间相关;program 标志将触发编码活动,编码过程完成后发出 debug 标志触发以速率 $p_2\rho$ 执行调试过程,调试过程执行后发出 ref 或者 pass 标志来决定是进入重构过程还是瞬时结束该模块开发。在调试活动中,缺陷数目 B 以一定速率 f_T 减少,直到满足约束条件瞬时终止或者被强制终止,缺陷修复过程与模块开发完成过程同步,它们的协作关系由◇标志。为每一个模块设置约束 R_i,⊥表示瞬时活动,通过发出此标志可以强制结束此次仿真过程。

3.3　随机进程代数仿真算法

基于随机进程代数模型,本文采用 Gillespie[15] 算法进行仿真。具体步骤如下:

a)初始化。设置系统的初始条件随机信息仿真参数、随机数生成器(服从指数分布的随机数)。

b)蒙特卡洛采样。在一定的时间间隔内,产生随机数来决定下一步活动时间,同时按一定速率产生此次活动的缺陷数量。

c)更新。加上前一步基础上随机产生的时间,并更新上一步所产生的缺陷数量。

d)迭代。回到步骤 a),直到超过仿真时间限制,后者满足退出条件。

通过每个模块度量的复杂性得到期望开发时间和期望缺陷数目,以这个作为指数分布的参数传入仿真模型。模块采用 TDD 方法后该模块的期望编码时间增加,增加的程度与模块的复杂度正相关。但是每一次 TDD 迭代会使该模块缺陷数目显著降低,仿真结果当满足约束 R_i 时,表示采用此种开发方法可行;如果无法满足约束 R_i,可依据设置的强制终止条件(最大执行时间)强行终止,表示此开发方法不能成功用于该模块的开发,这种情况暗示项目有开发延期风险。统计在仿真结果中成功和失败的次数,可以得出该模块开发风险概率和成功概率,有助于项目经理采取合理的决策。

3.4　随机进程代数 TDD 模块选取算法描述

对模块是否选择 TDD 的仿真算法伪码描述如下:

```
1. for C_i in C
2. CalulateComponentComplex(C_i) store[I,choice],
3. Map C_p ∝ γ
4. SimNoTDD(C_i,count,Stop):
5. Sim(CodeAct)
6. Sim(TestAct)
7. Sim(DebugAct)
8. if Satisfied R_i
9. Sum Ntime_i; success0 + +;cnt + +; if cnt < count continue Sim
10. else
11. if cnt < Stop
12. cnt + +;goto 5;
13. else
14. Ntime_i = +oo; cnt + +;if cnt < times continue Sim
15. choice; Sim(RefactorAct) goto 6
16. SimTDD(C_i,count,Stop):
17. Sim(PreTest)
18. Sim(CodeAct)
19. Sim(Debug);
20. if Satisfied R_i
21. Sum Time_i; success1 + +;cnt + +; if cnt < count continue Sim
22. else
23. if cnt < Stop
24. goto 18;cnt + +
25. else
26. Time_i = + oo; break;
27. choice; Sim(RefactorAct)
28. goto17.
29. if I0 > I1 store[I,0] else store[I,1]
30. next component;
31. output store
```

上述算法首先利用每个模块 UML 图中的用例图计算它的模块复杂度 C_P,对 C_P 进行处理得到随机进程代数仿真的参数 γ,该参数对于不同阶段的活动有不同的取值,考虑每个活动执行时间服从 γ 的负指数分布,γ 的倒数表示期望运行时间。对每一个模块进行 TDD 仿真和非 TDD 仿真,整个开发过程共有 $2^{C_{\mathrm{num}}}$ 种选择情况,最优 TDD 选取模块可以通过分析 store 表得出。

3.5 仿真实验与结果分析

采用本文的方法对某一实际项目进行仿真,该项目是一个多人协作的 Web 开发项目,项目小组采用敏捷开发方法,主要模块采用了 TDD 的开发思想,但是由于后期项目延期风险,TDD 方式没有得到始终贯彻执行,最终由于修复缺陷导致项目延期。该项目共 10 个主要模块,通过该项目的用例计算模块复杂度,从而估算各个模块期望开发时间与可能产生的缺陷数目(表 2)。

表 2　各个模块的仿真参数和约束

模块名称	用例个数	期望时间	总体仿真参数	时间约束	缺陷约束
M_1(auth)	4	9.0	0.11	10.3	1(7)
M_2(category)	3	3.2	0.31	3.6	1(4)
M_3(condition)	13	15	0.067	17.2	4(25)
M_4(project)	2	3	0.33	3.4	1(3)
M_5(view)	7	1.0	0.95	1.1	1(1)
M_6(stakeholder)	4	5.6	0.18	6.4	1(5)
M_7(priority)	4	5.9	0.17	6.7	1(5)
M_8(util)	1	3	0.33	3.4	1(3)
M_9(jdbc driver)	3	7.1	0.14	8.1	1(9)
M_{10}(ui)	3	4.5	0.22	5.1	1(3)
总计	44	57.3	——	65.3	13(65)

所有模块依据复杂度,以 M_5(view)模块为标准得到相应的期望开发时间(与复杂度正相关),每个模块的时间约束不超过模块期望开发时间的 115%。假设所有模块开发中产生的缺陷也与其复杂性正相关,模块缺陷数目约束表示为最终模块允许存在未修复的缺陷数目(模块预计产生的总缺陷数目 - 修复缺陷数目)。表 2 中第一列表示所有要开发的 10 个模块名称;第二列列出每个模块的用例个数;第三列为由模块复杂度得到的各模块期望开发时间;第四列为各模块总体仿真参数,为期望时间的倒数;第五、六列为时间约束和缺陷约束,缺陷约束表示模块最终允许存在的缺陷个数,括号中的数表示模块预计产生缺陷的数目。

假设在每一个模块的非 TDD 开发过程中,编码活动大致占 50% 时间,调试活动占 30%,测试活动 20%; TDD 相比较非 TDD 过程能够降低 18% 缺陷密度,增加 16% 的编码时间和测试时间,降低 25% 的调试时间[3],进入下一次迭代开发的概率随机,缺陷修复速率为修复个数/开发过程总体执行时间。non-TDD 与 TDD 的仿真参数如表 3、4 所示。表 3、4 第二列为编码过程参数,non-TDD 编码参数为模块期望编码时间的倒数,TDD 期望编码和期望测试时间为 non-TDD 的 1.16 倍。测试参数、调试参数计算方法与编码参数类似。表 3、4 的最后一列为缺陷修复速率参数,为修复的缺陷数目/整体开发时间。

表 3　non-TDD 模块约束和模块仿真参数

模块名称	编码过程参数($p'_1\lambda$)	调试过程参数($p'_2\rho$)	测试过程参数(δ')	缺陷修复参数(f_N)
M_1(auth)	0.22	0.37	0.55	0.59(6)
M_2(category)	0.62	1.04	1.56	0.83(3)
M_3(condition)	0.13	0.22	0.33	1.23(21)
M_4(project)	0.66	1.11	1.67	0.5(2)
M_5(view)	2.0	3.33	5.0	---(0)
M_6(stakeholder)	0.35	0.51	0.89	0.66(4)
M_7(priority)	0.33	0.56	0.84	0.62(5)
M_8(util)	0.66	1.11	1.67	0.63(2)
M_9(jdbc driver)	0.28	0.46	0.7	1(8)
M_{10}(ui)	0.44	0.74	1.11	0.4(2)

表 4　TDD 模块约束与模块仿真参数

模块名称	编码过程参数($P_1\lambda$)	调试过程参数($P_2\rho$)	测试过程参数(δ)	缺陷修复参数(f_T)
M_1(auth)	0.18	0.49	0.47	0.41(4)
M_2(category)	0.53	1.38	1.34	0.6(2)
M_3(condition)	0.14	0.20	0.37	0.94(14)
M_4(project)	0.56	1.49	1.43	0.32(1)
M_5(view)	1.71	4.43	4.31	---(0)
M_6(stakeholder)	0.30	0.68	0.76	0.49(3)
M_7(priority)	0.28	0.74	0.72	0.48(3)
M_8(util)	0.56	1.49	1.43	0.32(1)
M_9(jdbc driver)	0.24	0.61	0.60	0.8(6)
M_{10}(ui)	0.37	0.98	0.95	0.21(1)

本文以 M_{10} 为例将表中的仿真参数带入模型进行随机仿真实验,满足缺陷约束条件下开发时间仿真结果如图 6、7 所示。non-TDD 中 105 次仿真在满足时间约束和缺陷约束下共有 56 次成功,成功率为 53.3%,TDD 过程进行 115 次仿真,成功 72 次,成功率为 62.6%,如表 5 所示。

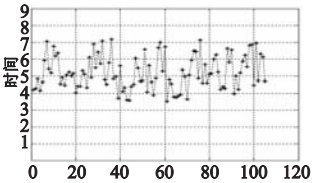


图6　non-TDD仿真结果

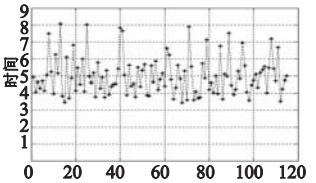


图7　TDD仿真结果

表 5　仿真结果

模块	TDD 成功率(成功/总数)	non-TDD 成功率(成功/总数)
M_1	0.616(98/159)	0.408(87/213)
M_2	0.422(54/128)	0.715(81/114)
M_3	0.564(88/156)	0.436(79/181)
M_4	0.402(80/199)	0.648(83/128)
M_5	1.0(---)	1.0(---)
M_6	0.62(75/121)	0.615(64/104)
M_7	0.69(80/116)	0.67(87/129)
M_8	0.402(80/199)	0.648(83/128)
M_9	0.829(131/158)	0.594(60/101)
M_{10}	0.626(72/115)	0.533(56/105)

其他模块的仿真结果如表 5 所示,从仿真结果分析大致可以看出,non-TDD 对于复杂度低的模块有更高的成功概率,如 M_2、M_4、M_8;TDD 对于复杂模块有较高的成功率;有些模块如 M_6、M_7 两者效果相差不多,这种情况优先选择 TDD 方法。M_5 由于复杂度低、开发时间短,无论哪种方式都可行。该项目实际开发中,由于进度原因对于 M_9、M_7、M_5 没有采用 TDD,结果为了解决 M_9 的缺陷花费了巨大的成本,导致项目延期。由此可见,本方法能够为项目管理人员选择 TDD 实施策略提供支持。

4 结束语

为了解决在软件项目开发中对哪些模块应用 TDD 方式开发能够获得更好的效果的实际问题,本文从 UML 图中的用例图出发,计算了每个模块的复杂度;采用随机进程代数对整个软件开发过程进行建模,利用模块的复杂度作为参数,利用随机进程代数(SPA)作为工具对整个开发过程进行仿真,最终能够为项目管理人员提供决策支持。本模型只是从模块自身的复杂度出发,并没有考虑人员变化、资源环境对项目中实施 TDD 的影响,同时为了避免状态爆炸,本仿真方法没有考虑模块间相互影响的因素,在未来的研究工作中可以在仿真模型中加入人员、环境的影响,并考虑模块相互依赖的关系,从而得到更全面的结果。

(下转第 2958 页)

可能地获取最优价值。

4 结束语

本文提出一种基于风险的需求优先级排序方法,建立了需求—任务双层模型,模型对需求的价值、成本、约束关系和风险因素进行了定义和描述,需求的成本及价值计算中包括了风险因素和需求间的关联关系,以费效比为依据进行需求优先级排序。方法的模拟仿真工具支持需求优先级的自动排序,支持项目的模拟执行,支持在指定约束下估算项目投入成本和产出价值的关系。模拟实验的结果表明该方法能够更好地获取价值,同时给出的需求优先级排序符合费效比原则,适用于强进度约束,开发自主性高、市场驱动的商业软件项目。该方法及其工具能协助项目经理进行项目开发内容的裁减和项目任务的计划,提高项目的价值和成功率。

目前,该方法的风险模型中,风险的可变因子只有发生概率和影响,因此无法描述如风险在不同时间点发生的影响差异,无法描述任务在项目尾期启动和项目早期启动的风险差异等,这在一定程度限制了实际风险情形向模型的转换。未来的工作将改进风险的描述,放宽假设条件,以更好地符合实际情形,提高方法在实际运用中的有效性。

参考文献:

[1] KARLSSON J. Software requirements prioritizing[C]//Proc of the 2nd Int'l Conference on Requirements Engineering (RE'96). Los Alamitos:IEEE Computer Society, 1996:110-116.

[2] RUHE G, GREER D. Quantitative studies in software release planning under risk and resource constraints[C]//Proc of the 2003 Int'l Symposium on Empirical Software Engineering (ISESE 2003). Los Alamitos: IEEE Computer Society, 2003:262-271.

[3] JUNG H W. Optimizing value and cost in requirements analysis[J]. IEEE Software,1998,15(4):74-78.

[4] CARLSHAMRE P, SANDAHL K, LINDVALL M, *et al*. An industrial survey of requirements interdependencies in software product release planning[C]//Proc of the 9th IEEE Int'l Conference on Requirements Engineering (RE 2001). Los Alamitos: IEEE Computer Society, 2001:84-93.

[5] GREER D, RUHE G. Software release planning: an evolutionary and iterative approach [J]. Information & Software Technology, 2004,46(4):243-253.

[6] SCHWALBE K. Information technology project management[M]. 4th ed. Beijing: China Machine Press, 2006.

[7] TRAN T, SHERIF J S. Quality function deployment (QFD): an effective technique for requirements acquisition and reuse[C]//Proc of the 2nd IEEE Software Engineering Standards Symposium Los Alamitos: IEEE Computer Society,1995:191-200.

[8] NGO-THE A N, RUHE G. Optimized resource allocation for software release planning[J]. IEEE Trans on Software Eng,2009,35(1): 109-123.

[9] LI Ming-shu, HUANG Meng, SHU Feng-di, *et al*. A risk-driven method for extreme programming release planning[C]//Proc of the 28th International Conference on Software Engineering. [S.l.]: ACM Press, 2006:423-430.

[10] 黄蒙,舒风笛,李明树.一种风险驱动的迭代开发需求优先级排序方法[J].软件学报,2006,17(12):2450-2460.

[11] XIE Li-zi. A project scheduling method based on human resource availability[C]//Proc of the 20th International Conferences on Software Engineering and Knowledge Engineering. San Francisco:Knowledge Systems Institute, 2008:161-166.

[12] WANG Qing, LI Ming-shu. Software process management: practices in China[C]//Proc of the International Software Process Workshop. Berlin: Springer-Verlag, 2005:317-331.

(上接第 2952 页)

参考文献:

[1] FUGGETTA A. Software process: a roadmap[C]//Proc of the Conference on the Future of Software Engineering. New York: ACM Press,2000: 25-34.

[2] JEFFRIES R,MELNIK G. TDD: the art of fearless programming[J]. IEEE Software,2007,24(3):24-30.

[3] TURNU I, MELIS M, CAU A, *et al*. Modeling and simulation of open source development using a agile practice[J]. Journal of Systems Architecture,2006,52(11):610-618.

[4] ERDOGMUS H, MAURIZIO M. On the effectiveness of test-first approach to programming[J]. IEEE Trans on Software Engineering, 2005,31(3):226-237.

[5] ZHANG Hong-yu. On the distribution of software faults[J]. IEEE Trans on Software Engineering,2008,34(2):301-302.

[6] KELLNER M I, MADACHY R J, RAFFO D M. Software process simulation modeling: Why? What? How? [J]. Journal of Systems and Software,1999,46:91-105.

[7] 刘夏.面向对象软件度量技术的研究与应用[D].济南:山东大学,2004.

[8] MA Yu-tao, HE Ke-qing, DU De-hui. A qualitative method for measuring the structural complexity of software metrics[C]//Proc of 12th Asia-Pacific Software Engineering Conference. Taiwan: [s. n.], 2005:257-263.

[9] HERMANNS H, HERZOG U, MERTSIOTAKIS V. Stochastic process algebra as a tool for performance and dependability modeling [C]//Proc of International Computer Performance and Dependability Symposium(IPDS'95).[S.l.]: IEEE CS Press,1995: 102-113.

[10] HILLSTON J. A compositional approach to performance modeling [D].[S.l.]: University of Edingburgh, 1994.

[11] TRIBASTONE M, DUGUID A, GILMORE S. The PEPA Eclipse plug-in[J]. ACM SIGMETRICS Performance Evaluation Review,2009,36(4):28-33.

[12] HERZOG U. Formal description, time and performance: a framework [C]//Informatik Fachberichte, vol.264, Entwurf und Betrieb verteilter Systeme. London: Springer-Verlag, 1990:172-190.

[13] MILNER R. A calculus of communication systems[M].[S.l.]: Springer-Verlag,1980.

[14] HAORE C A. Communicating sequence processes[J]. Commum ACM,1978,21(8):666-667.

[15] GILLESPIE D T. Exact stochastic simulation of coupled chemical reactions[J]. The Journal of Physical Chemistry,1977,81(25): 2340-2361.

[16] ZHAI Jian, LI Ming-shu,SU Feng, *et al*. Stochastic process algebra based software process simulation modeling[C]//Proc of the International Conference on Software Process. Heidelberg: Springer,2009: 136-147.

[17] YANG Qiu-song, LI Ming-shu,WANG Qing, *et al*. An algebra approach for managing inconsistencies in software process[C]//Proc of the 2007 International Conference on Software Process. Heidelberg: Springer-Verlag,2007:121-133.