# 基于事件约束的软件过程验证*

顾 庆+, 陈道蓄

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

## Event Constraint Based Software Process Validation

GU Qing+, CHEN Dao-Xu

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

+ Corresponding author: Phn: +86-25-83686550, Fax: +86-25-83300710, E-mail: guq@nju.edu.cn, http://cs.nju.edu.cn

Received 2004-06-15; Accepted 2005-06-10

**Gu Q, Chen DX. Event constraint based software process validation. *Journal of Software*, 2005,16(10): 1735−1742.** DOI: 10.1360/jos161735

**Abstract**: Software process is a human-centered system, with special characteristics in dynamic and continuous evolvement. The physical execution of a defined process will normally deviate from its process model. This paper uses E-CSPE (extended constraints on succeeding and proceeding events) constraints to carry out process validation and deviation measurement. The event constraints are defined based on the process model. The execution of a process instance is recorded down as an event sequence. The event sequence is analyzed to determine how much each event constraint defined is covered or violated. The result can be used to compute the EPD (event constraint based process difference metric) and EAD (event constraint based activity deviation metric) metrics. The EPD metric can reflect the difference between the process execution and its process model, while the EAD metric can provide some evidence for process evolvement.

**Key words**: software engineering; software process; software process improvement; process model; process validation; event constraint

摘 要: 软件过程是以人为中心的系统,其特点是动态性和不断演化.既定过程模型在实际执行时往往有所偏差.基于 E-CSPE(extended constraints on succeeding and proceeding events)约束实现过程验证和偏差测量.事件约束根据过程模型定义.过程实例执行被记录为事件序列.通过分析事件序列对事件约束的覆盖和违反结果,可以计算 EPD(event constraint based process difference metric)和 EAD(event constraint based activity deviation metric)指标.EPD 指标可以反映过程执行与过程模型的偏差,EAD 指标则为过程演化提供依据.

关键词: 软件工程;软件过程;软件过程改进;过程模型;过程验证;事件约束

中图法分类号: TP311 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2003AA113090, 2004AA112090 (国家高技术研究发展计划(863))

作者简介: 顾庆(1972－),男,江苏常州人,博士,副教授,主要研究领域为分布式语言和系统;陈道蓄(1947－),男,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算与并行处理.

提高软件开发的质量和生产率需要规范企业的软件过程.过程建模是规范软件过程的有效手段之一.当一个过程模型建立起来之后,接下来的问题是如何验证过程的实际执行符合过程模型的定义.与一般的工作流过程不同,软件过程涉及大量的人员协作.由于软件本身的特殊性,过程实施中容易出现变更.需要有一个有效的方法确保过程模型与过程执行的一致性,以充分发挥过程模型的作用.

规范软件过程已有相当充分的工作,典型的成果包括 CMM,CMMI,SPICE 和 RUP 等.一些较为成熟的软件企业纷纷规范并建立了自己的软件过程体系.但如何验证既定过程的实际实施目前尚有欠缺,过程验证多依赖于主观判断.这会导致软件人员对过程的实际执行效果缺乏信心,妨碍了过程模型的进一步应用和改进.

本文基于 E-CSPE 约束[1,2]提出一种过程验证方法,并对过程执行和过程模型之间的偏差进行度量.根据该方法,一方面可以发现并避免过程执行与过程模型定义的偏差;另一方面可据此确定过程模型与软件开发实际的距离,为过程演化提供依据.

## 1 基于事件的过程验证框架

### 1.1 过程事件和事件序列

软件过程的一次实例执行可以视为一个动作序列[3],其中每个动作代表一个活动或任务的执行.动作可能是一次应用程序执行,如编译一个模块的代码,也可能是一次人员活动,如召开一次评审会议.

过程事件代表一个可识别的和即时的动作.考虑到动作可能需要持续一段时间,对于事务性动作,可将动作的成功执行视为一个事件;对于持续时间很短的动作,如一次编译,也可视为一个事件;对于持续时间较长的非事务性动作,则可能对应多个事件,如召开评审会议至少需要对应“会议开始”和“会议结束”两个事件.

事件需要唯一的标识,针对过程验证的要求,事件至少应定义成一个三元组⟨Id,Type,Timestamp⟩.其中 Id 是记录事件时生成的唯一标识;Type 是事件的类型,对应相关的动作和动作的执行状态;Timestamp 是事件产生的时间,用于对过程事件排序.事件可能还需要包括其他属性,如相应动作的执行主体(人或应用程序),以及相关的工作产品(文档)等.

按照事件的定义,过程实例执行的动作序列可记录为一个事件序列.事件序列记录了过程一次执行的行为,可据此分析和测量同过程模型的符合程度.

### 1.2 过程事件和过程模型

为验证事件序列与过程模型的相关程度,需要将过程事件与过程模型关联起来.目前常用的过程定义模型有基于 Petri 网的过程模型[4]、基于任务流的过程模型[5]、基于有限状态机的过程模型[3]、基于过程式语言的过程模型[6],以及基于 ECA(event-condition-action)规则的过程模型[7]等.在这些模型中需要考虑过程事件的对应点,即确定模型中的事件源.

鉴于过程是由多个活动组成的有序集.只需找出各过程模型中活动的描述部分便不难确定相应的事件源.如 Petri 网模型的位置、托肯和变迁这 3 个基本描述单元中,变迁可作为事件源.任务流模型以活动结点为单位将过程描述成控制流图.活动结点又分复合活动和基本活动.复合活动可嵌套定义为子过程.这里不可再分的基本活动可作为事件源.有限状态机模型中状态之间的变迁可作为事件源.过程式语言中用于描述活动的代码段可作为事件源.对于 ECA 规则,其动作部分可作为事件源.ECA 规则本身存在事件的定义,但其事件分过程内部事件和外部事件.内部事件与活动的执行状态相关,符合过程事件的定义.外部事件则不属于过程事件的定义范畴.

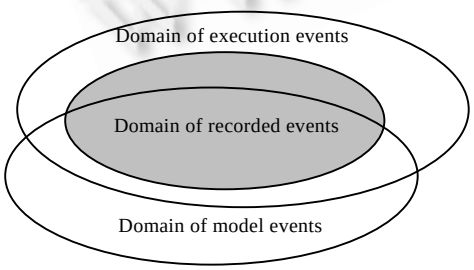



Fig.1 Three domains of process events

图 1 过程事件的 3 个域

基于过程模型的事件源可定义模型事件.据此,过程事件可分为 3 个不同的域[3]:模型事件域、执行事件域和记录事件域.3 个域之间的关系如图 1 所示.

其中,模型事件域是根据过程定义模型确定的事件集合.

执行事件域是过程一次实例执行中实际发生的事件集合.记录事件域是过程实例执行中记录下来的事件集合.

不难看出,存在过程事件,它属于模型事件域但不属于执行事件域.其原因,一是过程的一次实例执行一般不会覆盖过程定义的所有活动,二是过程执行与过程定义出现偏差.后者是过程验证需要分析的对象.偏差的原因可能有两个:一是过程的执行环境或目标发生变更,从而过程执行出现偏离,这构成过程演化的依据;二是由于疏忽或其他原因,本应执行的活动没有得到成功执行,这要求调整过程执行和监督策略,使既定过程得到充分贯彻.

为分析过程偏差,需要将过程执行事件记录下来从而形成记录事件域.显然,记录事件域是执行事件域的子域,可能存在执行事件未被记录下来的情况.如果未被记录的执行事件与过程验证无关,如不属于模型事件域且与模型事件无关,则可不予考虑.反之,若未被记录的执行事件与过程验证相关,则需调整事件记录策略,保证事件记录的充分性.

### 1.3 过程验证框架

本文讨论的过程验证框架主要分析并测量记录事件与模型事件之间的偏差情况.如图 2 所示,为过程验证框架的示意图,其中记录了事件组成事件序列.而模型事件根据过程模型的定义反映为一组事件约束.通过检查事件序列与事件约束集的符合程度并测量两者的相关度,可量化分析过程执行与过程模型的偏差,帮助过程管理人员制定决策.

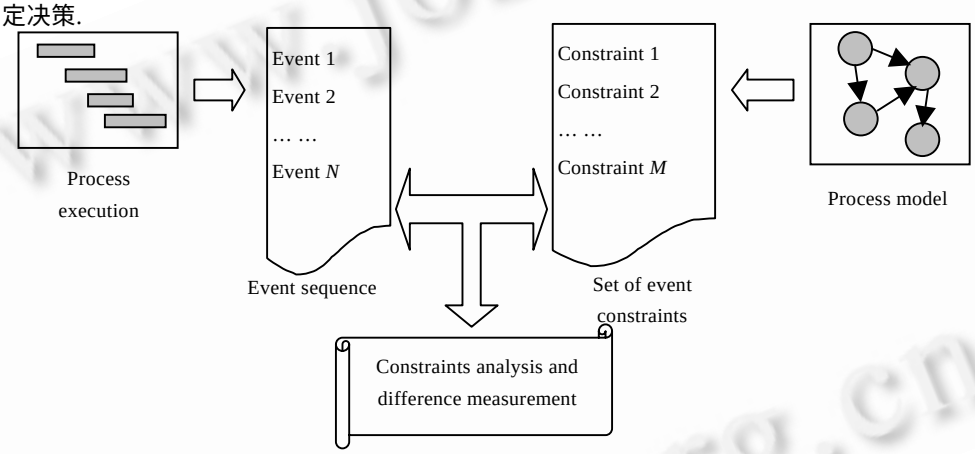



Fig.2 Process validation framework based on event constraints

图 2 基于事件约束的过程验证框架

过程验证框架中包括 3 方面操作:(1) 将过程执行记录为事件序列;(2) 根据过程模型推导事件约束集;(3) 应用事件约束集分析和测量事件序列与过程模型的偏差.本文以 E-CSPE 约束为基础探讨第 3 个功能操作.

## 2 基于 E-CSPE 约束的过程验证

### 2.1 E-CSPE约束

E-CSPE (extended CSPE)[1]是在 CSPE(constraints on succeeding and proceeding events)基础上提出的一个事件约束描述规则.E-CSPE 约束是指采用 E-CSPE 描述的事件约束.E-CSPE 约束在给定状态谓词下定义前后两个事件间的依赖关系以及这种关系的或然性.E-CSPE 约束的基本形式是 $op[[evt_1;\rightarrow evt_2]]_U C$.其中,$U$ 是指被测程序,这里可引申为过程模型;$evt_1$ 和 $evt_2$ 是前后相关的两个(类)记录事件;$C$ 是一个状态谓词,它决定 $evt_1$ 和 $evt_2$ 间的依赖关系是否存在;$op$ 规定依赖关系的或然性.$op$ 有 3 种取值,对应 3 类不同的 E-CSPE 约束,具体如下:

• $a[[evt_1;\rightarrow evt_2]]UC$:(always under condition),在过程模型 $U$ 的实例执行中,若条件 $C$ 成立,则 $evt_2$ 发生于 $evt_1$ 后总有效.

• $m[[evt_1;\rightarrow evt_2]]UC$:(must under condition),在过程模型 $U$ 的实例执行中,若条件 $C$ 成立,则 $evt_1$ 发生后,$evt_2$ 必发生.

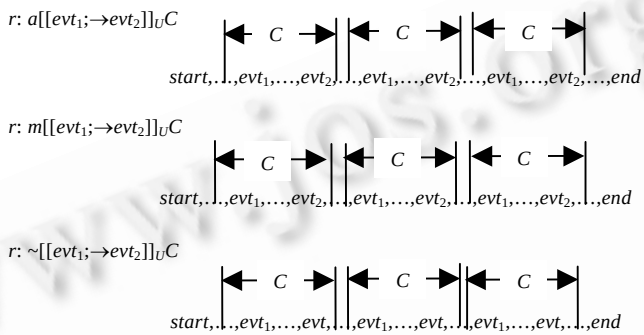
• ~[[$evt_1$;→$evt_2$]]$UC$:(never under condition),在过程模型 $U$ 的实例执行中,若条件 $C$ 成立,则 $evt_2$ 发生于 $evt_1$ 后总无效.

• 给定针对过程模型 $U$ 的 E-CSPE 约束 $r$ 和记录事件序列 $s$,$s$ 与 $r$ 之间可定义两种基本关系[1]:

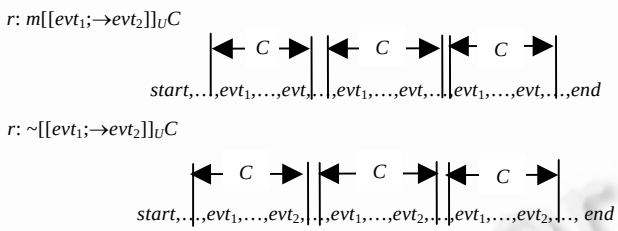
$s$ 覆盖 $r$.记为 $Con(s,r)$,表示 $s$ 中相邻的 $evt_1$ 和 $evt_2$ 事件对符合 $r$ 中的定义关系.

$s$ 违反 $r$.记为 $Vio(s,r)$,表示 $s$ 中相邻的 $evt_1$ 和 $evt_2$ 事件对违反 $r$ 中的定义关系.

如图 3 所示,为事件序列 $s$ 与约束 $r$ 的关系示意图.其中 *start* 和 *end* 为虚拟事件,标识序列 $s$ 的起始和终止.如图 3(a)所示,对于 $a$ 类和 $m$ 类约束,在条件 $C$ 的作用范围内,事件对〈$evt_1$,…,$evt_2$〉的出现覆盖约束 $r$;对于~类约束,$C$ 的作用范围内 $evt_1$ 出现后〈$evt_1$,…,$evt_2$〉未出现(图中为 $evt$)覆盖约束 $r$.如图 3(b)所示,只有 $m$ 类和~类约束可被违反.对于 $m$ 类约束,若 $C$ 作用范围内 $evt_1$ 出现后,〈$evt_1$,…,$evt_2$〉未出现违反约束 $r$;对于~类约束,若 $C$ 作用范围内〈$evt_1$,…,$evt_2$〉出现,则违反约束 $r$.




(a) $s$ conforms to $r$




(b) $s$ violates $r$

Fig.3 Illustration of the relation between constraint $r$ and event sequence $s$

图 3 约束 $r$ 与事件序列 $s$ 的关系示意图

由于软件开发常包含多次迭代,如参考生命周期模型中的渐增式模型和螺旋上升模型等,$s$ 中事件对〈$evt_1$,…,$evt_2$〉的出现次数可能不止一次.每次出现可能覆盖 $r$,也可能违反 $r$.据此可统计 $s$ 对约束 $r$ 的覆盖次数 $N^r_{con}$ 和违反次数 $N^r_{vio}$.

### 2.2 过程偏差指标

与程序测试不同,事件序列 $s$ 违反约束 $r$ 并不意味着过程执行出错,而是说明过程实例执行相对于过程模型定义发生了偏差.通过统计 $s$ 对各个约束 $r$ 的违反次数可量化反映过程执行的偏差程度.

为强化偏差程度的计算,可为不同类型的约束赋予不同的权值.如 $a$ 类约束中,事件 $evt_1$ 和 $evt_2$ 之间的依赖关系相对较弱,可赋予较低的权值,如"$W_a$=1".~类约束规定事件 $evt_1$ 发生后在给定条件 $C$ 下 $evt_2$ 对应的活动不应再被执行,$evt_2$ 的出现代表多余的活动,可对其赋予相对高的权值,如"$W_{\sim}$=2".$m$ 类约束规定事件 $evt_1$ 发生后在条件 $C$ 下 $evt_2$ 对应的活动必须被执行,$evt_2$ 没有出现代表既定过程没有得到遵从,可对其赋予最高的权值,如"$W_m$=4".

根据权值设定可计算过程偏差指标 EPD(event constraint based process difference metric).令 $Cset_a$,$Cset_m$ 和 $Cset_{\sim}$分别表示 3 类约束组成的约束集合,计算公式如下:

$$EPD=\frac{\sum_{r\in Cset_m}W_m\times N_{vio}^r+\sum_{r\in Cset_{\sim}}W_{\sim}\times N_{vio}^r}{\sum_{r\in Cset_a}W_a\times N_{con}^r+\sum_{r\in Cset_m}W_m\times N_{con}^r+\sum_{r\in Cset_{\sim}}W_{\sim}\times N_{con}^r} \tag{1}$$

计算 EPD 首先统计事件序列 $s$ 对各约束的违反次数,乘以对应的权值,累计得到过程偏差的绝对值.然后统计 $s$ 对各约束的覆盖次数,乘以权值并累计,得到过程一致程度的衡量值.以该值为分母归一化过程偏差值即得出 EPD 指标.

计算 EPD 需要处理一些特殊情况.如图 4 所示,考虑图 4(a),多个连续出现的 $evt_1$ 匹配一个 $evt_2$.这里,我们采用悲观的计算方法,即在合理范围内增大 EPD 的计算值.这是因为本应实施一次的活动被连续实施了多次,无论由于什么原因,已经代表过程执行出现了差错.3 种情况的处理如下:

- 考虑情况(a),对于 $a$ 类约束 $r$,$s$ 覆盖 $r$1 次;对于 $m$ 类约束 $r$,$s$ 覆盖 $r$1 次,但同时 $s$ 也违反 $r$1 次;对于~类约束 $r$,$s$ 违反 $r$2 次.但如果条件 $C$ 下 $evt_2$ 未出现,则 $s$ 覆盖 $r$2 次.
- 考虑情况(b),对于 $a$ 类约束 $r$,$s$ 覆盖 $r$1 次;对于 $m$ 类约束 $r$,$s$ 覆盖 $r$1 次;对于~类约束 $r$,$s$ 违反 $r$2 次.
- 考虑情况(c),由于可能有并发的情况存在.对于 $a$ 类约束 $r$,$s$ 覆盖 $r$2 次;对于 $m$ 类约束 $r$,$s$ 覆盖 $r$2 次;对于~类约束 $r$,$s$ 违反 $r$3 次.

显然,EPD 的计算取决于 Nrvio 和 Nrcon 的统计.令事件序列 $s$ 的长度为 $N$,对给定约束 $r$,统计 Nrvio 和 Nrcon 的计算复杂度为 $O(N^2)$.令约束 $r$ 的总数为 $M$,不难得出 EPD 的计算复杂度为 $O(MN^2)$.

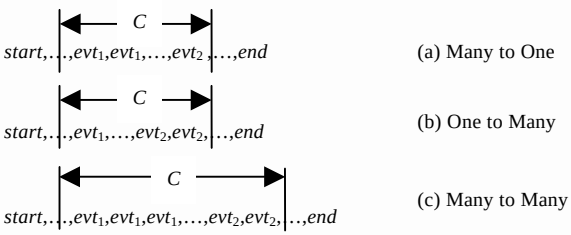



Fig.4 Special situations of event matching

图 4 事件匹配的特殊情况

### 2.3 活动偏离指标

EPD 指标仅考虑模型事件,且这些事件都参与约束处理.实际上,记录事件中还存在不属于模型事件域的执行事件以及未覆盖事件约束的模型事件.对这两类事件可作如下定义:

- 非模型事件:不属于模型事件域但属于执行事件域的记录事件.
- 非约束事件:属于模型事件域,分两种情况:一是没有覆盖任何约束的独立的记录事件,二是违反了某个事件约束的记录事件.对 $m$ 类约束,则指单独出现的 $evt_1$;对~类约束,则指不应出现的 $evt_2$.

出现非模型事件说明有新的过程需求要求新的活动;出现非约束事件意味着可能存在过程模型没有描述到的或者过时的事件约束.这些都可作为过程演化的依据.活动偏离指标 EAD(event constraint based activity deviation metric)用于衡量这两类事件带来的影响.

与计算 EPD 类似,为强调两类事件的差别,需要为其赋予不同的权值.如对于非模型事件,可赋予较大的权值,如 $W_{nm}$=4;而对于非约束事件,可赋予较小的权值,如 $W_{nc}$=2.在计算 EAD 时,事件序列中连续出现的非模型事件块或非约束事件块意味着过程活动执行出现了更大的偏离.这种偏离随着块长度的增加应该有着非线性的增长.本文参照文献[3]给出 EAD 的计算公式如下:

$$EAD=\frac{\sum_{i=1}^{N_{nm}^{B}}W_{nm}\times f(B_i)+\sum_{j=1}^{N_{nc}^{B}}W_{nc}\times f(B_j)}{W_{\max}\times L_s} \tag{2}$$

其中, $N_{nm}^{B}$ 和 $N_{nc}^{B}$ 分别为事件序列 $s$ 中非模型事件块和非约束事件块的个数;$B_i$ 和 $B_j$ 分别代表特定块的长度;$W_{\max}$ 为权值 $W_{nm}$ 和 $W_{nc}$ 中的较大者;$L_s$ 为序列 $s$ 的长度,即事件总数.函数 $f(B)$定义如下[3]:

$$f(B)=e^{k(B-1)} \tag{3}$$

其中,$k$ 为可调参数,要求"$k$>0.7"确保块长为 2 时函数 $f(B)$的值大于 2,从而实现非线性增长.($B$–1)保证当块长为 1 时,$f(B)$的值为 1.

计算 EAD 首先统计非模型事件块和非约束事件块导致的绝对偏离,然后由序列长度与最大权值的乘积将该偏离值归一化得到 EAD 指标.显然,EAD 的计算复杂度取决于非模型事件和非约束事件的识别.若序列 $s$ 长度为 $N$,约束集规模为 $M$,则识别一个非模型事件的复杂度为 $O(M)$;识别一个非约束事件的复杂度为 $O(MN)$.由此不难得出 EAD 的计算复杂度为 $O(MN^2)$.

## 3 实例研究

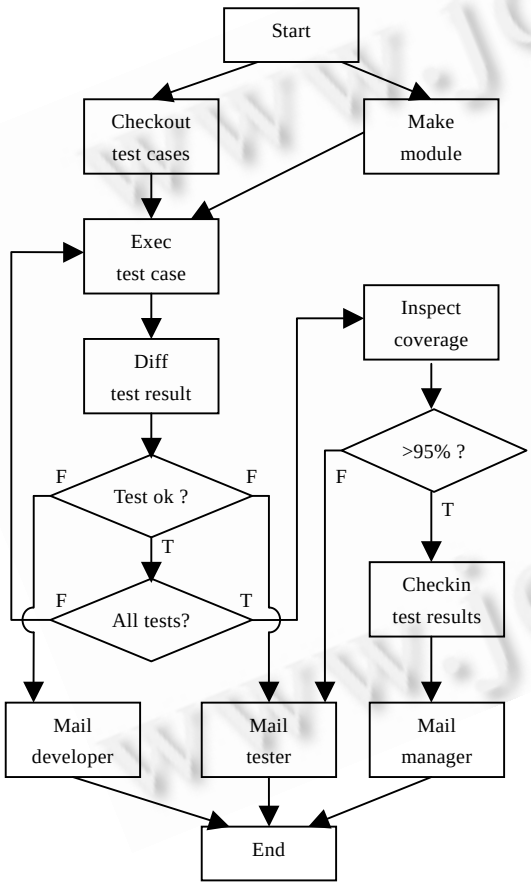



Fig.5 Task flow diagram of unit testing process

图 5 单元测试过程的任务流图

为了有效说明过程偏差指标和活动偏离指标,本文结合 ISPW 6 中的单元测试过程实例[8]介绍两个指标的计算.如图 5 所示,为单元测试过程的任务流图表示.该过程的执行角色为开发者和测试人员;处理的工作文档包括测试用例、单元模块源代码、单元模块执行程序以及测试结果报告.过程的起始条件是单元模块因需求变更而被修改;终止条件是选定的测试用例集执行完毕,根据测试覆盖度是否达到 95%及测试是否出错来报告测试结果.为简化起见,图中未画出角色、文档等内容.

在如图 5 所示的过程中,除了虚拟活动 *start* 和 *end* 以外,主要有以下几类活动:

- checkout:从配置库中提取测试用例集;
- make:按测试要求编译单元模块;
- exec:执行一个测试用例;
- diff:比较单元输出和用例预期结果;
- inspect:检查测试覆盖度;
- checkin:单元测试结果入配置库;
- mail-d:将测试问题反馈给开发者;
- mail-t:将测试问题反馈给测试人员;
- mail-m:将测试结果汇报给管理者.

假设每个活动都即时结束.令事件标识为事件的序号,时间戳已用于事件排序,则事件可以简单地表示为活动名.序列中前后同名的事件为同类事件.

根据单元测试过程的流图表示,可列出如下所示的 E-CSPE 约束:

r1. a[[checkout→exec]](true)

r2. a[[make→exec]](true)

r3. m[[exec→diff]](true)

r4. m[[diff→mail-d]](test failed because of code problem)

r5. m[[diff→mail-t]](test failed because of test problem)

r6. ~[[diff→mail-d]](test ok or test failed because of test problem)

r7. ~[[diff→mail-t]](test ok or test failed because of code problem)

r8. m[[diff→exec]](test ok and more test cases exist)

r9. m[[diff→inspect]](test ok and all test cases executed)

r10. ~[[diff→exec]](test failed or all test cases executed)

r11. ~[[diff→inspect]](test failed or more test cases exist)

r12. m[[inspect→checkin]](test coverage more than 95%)

r13. m[[inspect→mail-t]](test coverage no more than 95%)

r14. ~[[inspect→checkin]](test coverage no more than 95%)

r15. ~[[inspect→mail-t]](test coverage more than 95%)

r16. m[[checkin→mail-m]](true)

表 1 列出了在实验室环境下组织的单元测试过程 5 次实例执行的情况.5 位测试者自己编程序并完成 3 个用例的单元测试.其中第 1 次实例执行用于演示,测试过程成功结束且完全符合过程模型.表中问题程度为与软件从业人员讨论得出的各实例执行的问题严重情况.“diff-F”表示单元输出与用例预期不符.“check”是模型中没有描述的事件,指人工检查单元测试的输出.

**Table 1** 6 instance’s execution of the unit testing process

表 **1** 单元测试过程的 6 次实例执行

| No. | Recorded event sequences | Severity |
|---|---|---|
| 1 | start, checkout, make, exec, diff, exec, diff, exec, diff, inspect, checkin, mail-m, end | Normal |
| 2 | start, make, make, checkout, make, exec, diff-F, exec, mail-d, mail-t, end | Bad |
| 3 | start, checkout, make, exec, diff, exec, diff, exec, inspect, checkin, mail-m, end | Bad |
| 4 | start, checkout, make, exec, diff, exec, diff, exec, diff, checkin, mail-m, end | Severe |
| 5 | start, checkout, make, exec, diff-F, **check,** diff, exec, diff, exec, diff, inspect, checkin, mail-m, end | Light |
| 6 | start, checkout, make, exec, diff, **check**, exec, diff, **check, check**, mail-t, end | Recommend |

表 2 给出了针对上述 6 个事件序列指标 EPD 和 EAD 的计算结果.其中 $N^r_{con}$ 为事件序列对各约束的覆盖次数; $N^r_{vio}$ 为序列对各约束的违反次数;$N_{nm}$为序列中非模型事件的个数;$N_{nc}$为序列中非约束事件的个数;$C$为 EAD 的界值,按式 $0.1e^k$ 计算[3];EPD 的界值为 0.2.

**Table 2** Calculation of EPD and EAD metrics for each event sequence

表 **2** 各事件序列的 EPD 和 EAD 指标计算

| No. | EPD ($W_a$=1, $W_-$=2, $W_m$=4) | | | | EAD ($W_{nm}$=4, $W_{nc}$=2, $k$=1) | | | | EAD ($W_{nm}$=4, $W_{nc}$=2, $k$=2) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $N^r_{con}$ | $N^r_{vio}$ | Violate | EPD | $N_{nm}$ | $N_{nc}$ | EAD | C | $N_{nm}$ | $N_{nc}$ | EAD | C |
| 1 | 20 | 0 | None | 0 | 0 | 0 | 0 | 0.27 | 0 | 0 | 0 | 0.74 |
| 2 | 5 | 3 | r3,r7,r10 | 0.67 | 0 | 4 | 0.26 | 0.27 | 0 | 4 | 0.52 | 0.74 |
| 3 | 14 | 2 | r3, r11 | 0.16 | 0 | 2 | 0.14 | 0.27 | 0 | 2 | 0.37 | 0.74 |
| 4 | 17 | 1 | r9 | 0.09 | 0 | 1 | 0.05 | 0.27 | 0 | 1 | 0.05 | 0.74 |
| 5 | 23 | 1 | r5 | 0.07 | 1 | 1 | 0.12 | 0.27 | 1 | 1 | 0.12 | 0.74 |
| 6 | 10 | 2 | r7, r8 | 0.25 | 3 | 2 | 0.51 | 0.27 | 3 | 2 | 1.21 | 0.74 |

根据表 2 的计算结果,参照表 1 给出的问题程度,可导出以下认识:

- EPD 可有效反映过程执行的偏差程度,如事件序列 2,3 和 6;这方面 EAD 可起辅助作用.与之对照,EAD 在反映过程演化的需求上有较大的借鉴意义,如事件序列 6;EPD 在此可用作参考.
- 按类别设定事件约束的权值有时不能准确反映过程偏差带来的问题的严重程度.比如,问题严重的序列 4,其 EAD 和 EPD 值都很低.若考虑过程语义,则约束 r9 应赋予较高的权值.
- 与事件约束的权值设置类似,一些事件应作为关键事件设定较高的权值,如 inspect 事件.从而使 EAD 值的计算能够反映过程执行的问题程度.

## 4 相关工作比较

文献[9]利用自动机模型定义以人为中心的系统(如软件过程)及其过程支持系统.再以此为基础定义行为偏差和状态不一致问题.利用事件序列反映行为偏差,描述了过程偏差的概念.其重点是过程支持系统如何完整并一致地支持以人为中心的系统行为.

文献[3]基于自然语言识别方法比较过程模型和过程执行产生的事件序列.提出计算过程偏差的线性距离和非线性距离两个指标.计算依据是字符(即事件)的插入和删除操作.提出了过程验证框架,以自动机模型为基础量化过程执行与过程模型的背离.文献[3]将语法检查概念直接引入过程验证领域,没有区别过程偏差和过程演化需求.本文的工作是其有效扩展.

文献[4,5]分别基于 Petri 网和任务流提出过程验证的概念,重点在证明过程模型本身的正确性,适用于以工作流为基础的全自动过程支持系统.文献[10]提出面向对象的过程描述语言 $E^3$,以实现软件过程的改进和精化,通过过程组装和重用,解决过程模型的不完整和不一致问题.

## 5 结 论

软件过程是以人为中心的系统,其突出特点是动态性和不断演化.既定过程模型在实际执行时往往有所偏差.本文基于 E-CSPE 约束提出过程验证框架.首先根据过程模型定义事件约束.然后将过程实例执行记录为事件序列.最后通过分析事件序列对事件约束的覆盖和违反情况计算 EPD 和 EAD 两个指标.EPD 指标可反映过程执行同过程模型的偏差;EAD 指标则为过程演化提供了一定的依据.

在此基础上,进一步的工作有两个方面:一是根据各种过程模型推导事件约束集;二是当存在过程演化的需求时,根据事件序列推演新的规则,为过程模型的改进提供参照.

致谢 感谢参与实验的各位同学;感谢评审专家的宝贵意见.

**References**:

[1] Gu Q, Chen DX, Yu M, Xie L, Sun ZX. Validation test of distributed program based on events sequencing constraints. Journal of Software, 2000,11(8):1035−1040 (in Chinese with English abstract).

[2] Gu Q, Chen ZY, Chen DX, Xie L. Consistency checks of E-CSPE constraints. Chinese Journal of Computers, 2003,26(11): 1568−1574 (in Chinese with English abstract).

[3] Cook JE, Wolf AL. Software process validation: Quantitatively measuring the correspondence of a process to a model. ACM Trans. on Software Engineering and Methodology, 1999,8(2):147−176.

[4] van der Aalst WMP. Workflow verification: Finding control-flow errors using Petri-net-based techniques. In: van der Aalst W, Desel J, Oberweis A, eds. Business Process Management: Models, Techniques, and Empirical Studies, LNCS 1806, London: Springer-Verlag, 2000. 161−183.

[5] Sadiq W, Orlowska ME. Analyzing process models using graph reduction techniques. Information Systems, 2000,25(2):117−134.

[6] Sutton SM, Heimbigner JRD, Osterweil LJ. APPL/A: A language for software process programming. ACM Trans. on Software Engineering and Methodology, 1995,4(3):221−286.

[7] Ceri S, Grefen P, S'anchez G. WIDE—A distributed architecture for workflow management. In: Sipple RS, ed. Proc. of the 7th Int'l Workshop on Research Issues in Data Engineering. Los Alamitos: IEEE Computer Society Press, 1997. 76−79.

[8] Kellner MI, Feiler PH, Finkelstein A, Katayama T, Osterweil LJ, Penedo MH, Rombach HD. Software process modeling example problem. In: MacCallum AC, ed. Proc. of the 6th Int'l Software Process Workshop. Los Alamitos: IEEE Computer Society Press, 1990. 19−29.

[9] Cugola G, Nitto ED, Fuggetta A, Ghezzi C. A framework for formalizing inconsistencies and deviations in human-centered systems. ACM Trans. on Software Engineering Methodology, 1996,5(3):191−230.

[10] Jaccheri ML, Picco GP, Lago P. Eliciting software process models with the $E^3$ language. ACM Trans. on Software Engineering and Methodology, 1998,7(4):368−410.

附中文参考文献:

[1] 顾庆,陈道蓄,于勐,谢立,孙钟秀.基于事件约束的分布式程序正确性测试.软件学报,2000,11(8):1035−1040.

[2] 顾庆,陈宗岳,陈道蓄,谢立.E-CSPE 约束的一致性判定.计算机学报,2003,26(11):1568−1574.