软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(6):2844−2862 [doi: 10.13328/j.cnki.jos.006907]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于模板匹配的 BPEL 程序故障修复及优化技术*

孙昌爱, 吴思懿, 张守峰, 付 安

(北京科技大学 计算机与通信工程学院, 北京 100083)

通信作者: 孙昌爱, E-mail: casun@ustb.edu.cn

摘 要: BPEL (business process execution language) 是一种可执行的 Web 服务组合语言. 与传统程序相比, BPEL 程序在编程模型、执行方式等方面存在较大差异. 这些新特点使得如何定位并修改测试阶段发现的 BPEL 程序故障成为挑战, 面向传统软件的故障修复技术难以直接应用于 BPEL 程序. 从变异分析角度出发, 提出一种基于模板匹配的 BPEL 程序故障修复方法 BPELRepair. 为了克服基于变异分析的故障修复技术计算开销高的缺点, 从补丁生成、测试用例选择以及终止条件 3 个角度提出多种优化策略. 开发一个 BPEL 故障修复支持工具, 提高故障修复的自动化程度与效率. 采用经验研究的方式, 评估所提故障修复技术及优化策略的有效性. 实验结果表明, 所提故障修复方法能够成功修复约 53% 的 BPEL 程序故障; 所提优化策略能够显著降低搜索匹配、补丁程序验证、测试用例执行与故障修复等方面的开销.

关键词: 程序调试; 故障修复; Web 服务; 服务组合; BPEL 程序

中图法分类号: TP311

中文引用格式: 孙昌爱, 吴思懿, 张守峰, 付安. 基于模板匹配的BPEL程序故障修复及优化技术. 软件学报, 2024, 35(6): 2844–2862. http://www.jos.org.cn/1000-9825/6907.htm

英文引用格式: Sun CA, Wu SY, Zhang SF, Fu A. Template Matching-based Fault Fixing Technique for BPEL Programs and Its Optimization. Ruan Jian Xue Bao/Journal of Software, 2024, 35(6): 2844–2862 (in Chinese). http://www.jos.org.cn/1000-9825/6907.htm

Template Matching-based Fault Fixing Technique for BPEL Programs and Its Optimization

SUN Chang-Ai, WU Si-Yi, ZHANG Shou-Feng, FU An

(School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract: Business process execution language (BPEL) is an executable web service composition language. Compared with traditional programs, BPEL programs are significantly different in terms of programming models and execution modes. These new features make it challenging to locate and fix faults of BPEL programs detected during the testing process. In addition, fault fixing techniques developed for traditional software cannot be used for BPEL programs directly. This study proposes a fault fixing technique for BPEL programs based on template matching, namely BPELRepair from the perspective of mutation analysis. In order to overcome the high computational overhead of the mutation analysis-based fault fixing technique, a set of optimization strategies are proposed from three perspectives, namely patch generation, test case selection, and termination condition. A supporting tool is developed to improve the automation and efficiency of fault fixing for BPEL programs. An empirical study is used to evaluate the effectiveness of the proposed fault fixing technique and optimization strategies. The experimental results show that the proposed technique can successfully fix about 53% of faults of BPEL programs, and the proposed optimization strategies can significantly reduce the overhead in terms of search matching, patch program verification, test case execution, and fault fixing.

Key words: program debugging; fault fixing; Web service; service composition; business process execution language (BPEL) program

近年来, 面向服务的架构 (service oriented architecture, SOA) 已经成为主流的软件开发范型[1,2]. Web 服务是 SOA 架构下软件系统的基本组成单元[3], 通常由不同的机构或组织完成. BPEL (business process execution

* 基金项目: 国家自然科学基金 (61872039, 62272037); 北京市自然科学基金 (4162040); 航空科学基金 (2016ZD74004)
收稿时间: 2022-09-20; 修改时间: 2022-11-17; 采用时间: 2023-01-19; jos 在线出版时间: 2023-07-05
CNKI 网络首发时间: 2023-07-06

language)[3]是一种广泛采纳的可执行的服务组装语言, 从工作流的角度协调参与业务流程的外部 Web 服务, 而实现的流程本身可以看作一个新的 Web 服务. 以这样的方式, BPEL 可以支持不同粒度、复杂的服务组合. 与传统的 C、C++、Java 程序相比, BPEL 程序具有以下特点[3].

(1) BPEL 程序支持显式的 Web 服务组装方式, 而传统程序侧重于各种计算.

(2) BPEL 程序采用伙伴链接活动调用 Web 服务, 这些服务可能由不同的编程语言实现.

(3) BPEL 程序呈现为 XML 文件, 不同于传统应用程序.

(4) BPEL 程序用带有 link 的 flow 活动来支持并发同步机制, 传统程序通过多线程的方式实现并发机制.

程序调试[4]旨在排除软件测试阶段检测的故障, 包括故障定位和故障修复. BPEL 程序的特点使得其调试面临新挑战[5]. 具体来说: (1) BPEL 程序侧重服务之间的交互与协调, 而传统程序侧重于数据的操作与存储, 导致在编程模型、语法格式、执行方式等方面区别于传统程序; (2) BPEL 程序以 XML 格式呈现, 而针对 XML 程序调试的研究工作并不多见; (3) BPEL 程序将结构信息和交互信息同时嵌入到 XML 程序中. 上述 BPEL 程序的新特点使得面向传统程序的调试技术难以直接应用于 BPEL 程序. 为此, 人们提出了一些面向 BPEL 程序的故障定位技术, 主要包括: 1) 基于块结构的 BPEL 程序故障定位技术[6]: 利用程序频谱应用于 BPEL 程序块结构, 计算每个语句块的故障怀疑度; 2) 基于谓词切换与程序切片相结合的 BPEL 程序故障定位技术[7]: 采用程序切片技术分析与关键谓词切换相结合法给方式定位 BPEL 程序的故障片段; 3) 基于变异分析的 BPEL 程序故障定位技术[8]: 通过变异获得 BPEL 故障程序所有变异体, 统计每个语句块对应变异体集的执行信息并结合怀疑度计算公式计算语句块故障可能性. 上述技术中, 基于变异分析的 BPEL 程序故障定位技术具有较好的定位精度与定位效率, 因此本文采用该技术确定故障语句块位置. 现有 BPEL 程序故障修复方面的主要工作包括: 1) 使用正确 BPEL 程序代替存在故障的 BPEL 程序[9]. 此类方法假设存在正确的 BPEL 程序版本; 2) 基于 BPEL 语言异常处理机制的故障修复技术[10,11]: 人为预先定义程序故障时执行的特定操作, 包括终止、重做、回滚、警报、替换以及其他补偿操作. 此类方法提供了一种增强 BPEL 程序鲁棒性的机制, 但是需要人工设计相关的修复操作.

本文提出了一种基于模板匹配的 BPEL 程序故障修复方法. 首先将语句块类型信息与变异算子组合构造修复模板. 之后, 利用 BPEL 程序的故障语句块类型信息选取对应的修复模板, 结合修复模板中的变异算子对故障语句块执行变异操作生成候选补丁程序, 最后执行测试用例集验证补丁程序的正确性. 为了解决补丁生成、补丁验证等过程开销高的问题, 本文从补丁生成、测试用例选择及终止条件 3 个角度提出了 7 种优化策略. 开发了一个 BPEL 故障修复支持工具, 提高故障修复的自动化程度与效率. 采用 6 个程序实例的 403 个故障版本, 验证本文所提故障修复技术及优化策略的有效性.

本文第 1 节介绍基础知识, 包括 BPEL 程序、变异分析和 BPEL 变异算子. 第 2 节介绍基于模板匹配的故障修复技术及优化策略. 第 3 节对所提技术与优化策略的有效性进行实验评估. 第 4 节介绍基于搜索的故障修复与 BPEL 程序的故障定位研究现状. 最后总结全文.

1 背景知识

本节介绍 BPEL 程序、变异分析和 BPEL 程序的变异算子.

1.1 BPEL 程序

BPEL 程序以 XML 的格式呈现, 显式地组合 Web 服务并实现具体的业务逻辑. 具体说来, 通过 BPEL 活动 (activity) 将流程中的每个组件依据业务逻辑进行连接, 形成完整的业务流程. 主要的组件类型如下.

- 伙伴链接: 描述 BPEL 程序与关联 Web 服务间关系, 包括调用外部 Web 服务或自身对外提供的 Web 服务.
- 变量: 定义 BPEL 程序使用的数据, 负责存储与 Web 服务交互过程中的输入和输出数据.
- 补偿处理: 如果 BPEL 程序中的某一个活动发生异常, 撤销该活动之前的活动任务, 实现业务回滚.
- 错误处理: 通过定义适当的错误处理机制, 捕获到异常并作处理, 保证业务流程正常进行.

BPEL 程序的活动负责业务流程的具体执行, 分为基本活动 (basic activity) 和结构化活动 (structured activity), 见表 1. 其中, 基本活动描述业务流程的基本简单步骤, 而结构化活动用于管理业务流程的控制流逻辑.

表 1 BPEL 程序主要活动描述

Category	Activity	Description
Basic activity	Receive	The beginning of a business process, responsible for receiving the request messages sent by a client
	Reply	The end of the business process, responsible for returning the results of the business operation to the client
	Assign	Assign a parameter under the <from> variable to a parameter under the <to> variable
	Invoke	Invoke a service corresponding to a partner link and collect the result returned by the service
Structured activity	Sequence	Execute each activity within a Sequence in a given order
	If	Determine whether certain conditions are met to perform a particular operation
	While (RepeatUntil)	Determine whether certain conditions are met to enter a loop in an activity
	Flow	Define the activities that need to be executed in parallel
	Scope	Define the action scope of an internal activity and provides a common component for the activities within it

1.2 变异分析

变异分析 (也称变异测试) 是一种基于缺陷的软件测试技术[12,13], 广泛用于评估测试用例集的充分性以及测试技术的有效性. 变异分析通过模拟编程人员在软件开发过程中可能出现的一些错误植入故障, 执行测试用例集, 判断测试用例集揭示这些故障的能力. 测试用例集的故障检测能力可以通过变异得分[14]来衡量, 即测试用例集能够杀死的非等价变异体 (语义上与原程序不一致的变异体) 占所有的非等价变异体的比例.

变异分析基于两个重要的假设[13]: 熟练程序员假设 (熟练程序员编写出的缺陷代码与正确代码之间只有细微差别, 仅通过小幅度代码修改即可更正错误) 和耦合效应假设 (若某个测试用例可以检测出简单故障, 那么该测试用例也有潜力检测出更复杂的故障).

传统变异分析流程如下[15]: (1) 根据程序语言特征设计一组变异算子; (2) 在原程序上应用变异算子生成大量变异体; (3) 从变异体集合中识别出等价变异体 (语义上与原程序一致的变异体) 并筛除; (4) 测试剩余变异体, 若执行结果与原程序结果不一致, 则称该变异体被杀死; 否则, 变异体存活.

1.3 BPEL 变异算子

变异算子定义了将原程序转变为变异体的转换规则, 是实施变异分析的关键. 表 2 总结了常见的 BPEL 程序算子[16], 包括标识符替换、表达式变异、活动变异、异常与事件变异这 4 类.

表 2 BPEL 程序变异算子描述

Category	Mutation operator	Description
Identifier substitute mutation	ISV	Substitute a variable identifier with another one of the same type
Expression mutation	EAA	Substitute an arithmetic operator (+ , – , * , /) with another operator
	EEU	Remove the unary subtraction operator from an expression
	ERR	Substitute a relational operator (< , > , <= , >= , = , ! =) with another one
	ELL	Substitute a logical operator (and , or) with another one
	ECC	Substitute a path operator (/ , //) with another one
	ECN	Increase or decrease the value of a numeric constant, add or remove a number
	EMD	Set the duration to 0, or reduce the duration by half
	EMF	Set the cut-off time to 0, or reduce the cut-off time by half
	EIN	Insert XPath format “not” into a logical expression
	EIU	Insert XPath format unary subtraction into an arithmetic expression
	EAP	Substitute an expression with its absolute value
	EAN	Substitute an expression with its negative absolute value
Override criterion mutation	CFA	Replace an activity with an exit activity
	CDE	Apply decision coverage criterion to a program
	CCO	Apply condition coverage criterion to a program
	CDC	Apply decision/condition coverage criterion to a program

表 2 BPEL 程序变异算子描述 (续)

Category	Mutation operator	Description
Exception and event mutation	XMF	Remove the “catch” or “catchall” elements from error handlers
	XMC	Remove the definition of a compensation handler
	XMT	Remove the definition of a termination handler
	XTF	Substitute the thrown fault with a throw activity
	XER	Remove a rethrow activity
	XEE	Remove the onEvent element from an event handler
Concurrency-independent activity mutation	AEL	Remove an activity
	AIE	Remove the “elseIf” or “else” element from an if activity
	AWR	Substitute a while activity with a repeatUntil activity
	AJC	Remove the joinCondition element
	ASI	Exchange two child activities within a sequence activity
	APM	Remove the onMessage element of a pick activity
	APA	Remove the onAlarm element that can be associated to a pick activity
Concurrency-related activity mutation	ACI	Change the value of createInstance attribute from “yes” to “no”
	AFP	Change a sequential forEach activity to parallel
	ASF	Substitute a sequence activity with a flow activity
	AIS	Change the isolated attribute of a scope to “no”

2 基于模板匹配的 BPEL 程序故障修复技术

本节提出一种基于模板匹配的 BPEL 程序故障修复技术, 首先介绍该技术的基本原理, 包括方法框架、故障修复规则与算法等, 然后讨论优化策略, 最后用一个例子演示方法的使用.

2.1 基于模板匹配的故障修复方法框架

通常说来, 基于变异分析的故障修复技术的基本思想是: 通过对故障程序执行变异操作生成可能正确的程序. 该方法基于如下假设[17]: 如果一个真实的错误来自于一个正确程序的微小修改, 那么一个故障程序的修改则有可能产生正确的修复. 图 1 示意一个故障程序 (变异体) 及其正确的修复版本. 故障程序可以看作是 BPEL 变异算子 ISV (相同类型的变量标识符替换) 生成的变异体, 即将对变量 *b* 的赋值错误写成对变量 *a* 的赋值; 而故障修复则可以看成是上述变异分析的逆过程, 即将变量 *a* 修改为变量 *b*. BPEL 程序由若干个语句块组成, 每个语句块属于特定的语句块类型. 而不同的语句块类型仅能适用特定类型的 BPEL 变异算子. 例如, 由于图 1 示例的程序不包含 sequence, 因此 BPEL 变异算子 ASF (将 sequence 活动替换为 flow 活动) 不适用. 利用全部变异算子对故障语句块进行故障修复时, 仅有少数几个变异算子是有效并能产生候选补丁程序. 因此, 通过分析语句块类型与每种 BPEL 变异算子的匹配关系, 即每种语句块类型能够适用哪些变异算子, 从而构造修复模板; 然后利用这些修复模板, 可以避免不必要的变异算子搜索匹配过程.

```
1  <bpel:assign validate="no" name="AssignParam">
2    <bpel:copy>
3      <bpel:from>
4        <bpel:literal>
5          <impl:add xmlns:impl="http://add.calc.example.com">
6            <impl:a>0.0</impl:a>
7            <impl:a>0.0</impl:b>
8          </impl:add>
9        </bpel:literal>
10     </bpel:from>
11     <bpel:to variable="addRequest" part="parameters">
12     </bpel:to>
13   </bpel:copy>
14 </bpel:assign>
```

(a) 故障程序

```
1  <bpel:assign validate="no" name="AssignParam">
2    <bpel:copy>
3      <bpel:from>
4        <bpel:literal>
5          <impl:add xmlns:impl="http://add.calc.example.com">
6            <impl:a>0.0</impl:a>
7            <impl:b>0.0</impl:b>
8          </impl:add>
9        </bpel:literal>
10     </bpel:from>
11     <bpel:to variable="addRequest" part="parameters">
12     </bpel:to>
13   </bpel:copy>
14 </bpel:assign>
```

(b) 修复程序

图 1 研究动机示例

从变异分析的角度, 我们提出了基于模板匹配的 BPEL 程序故障修复技术, 简称为 BPELRepair. 图 2 示意了 BPELRepair 的方法原理, 包含如下 4 个步骤.

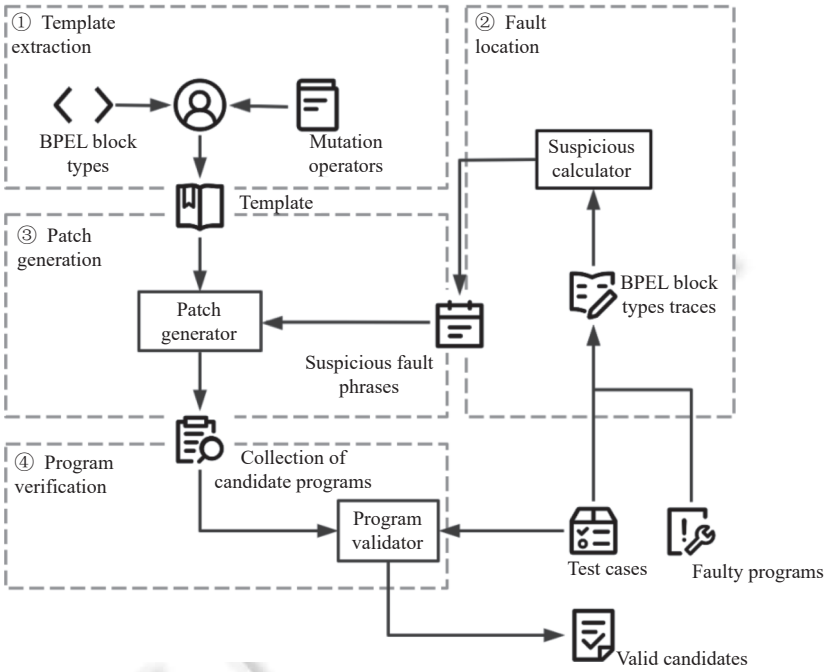

图 2 BPELRepair 的方法原理图

(1) 模板抽取: 在分析 BPEL 程序的特点的基础上, 将语句块类型和与其适用的所有变异算子组合得到 BPEL 故障修复模板.

(2) 故障定位: 使用测试用例集执行故障程序, 收集执行结果与执行轨迹, 利用这些信息并结合怀疑度计算公式 (本文选用 Tarantula[8]、DStar[8]、CBI[18]和 Ochiai[18]) 计算每个语句块的故障怀疑度, 最后依照故障怀疑度进行排序.

(3) 补丁生成: 补丁生成器 (patch generator) 利用故障定位阶段的语句块排序结果, 依次针对可疑语句块展开修复. 首先, 确定可疑语句块及其嵌套的语句块的类型, 找到这些语句块类型所对应的修复模板; 然后, 获取修复模板内的修复操作集合; 最后, 对相关语句块的代码进行搜索匹配以确定修复操作的适用性, 使用适用的修复操作对该语句块执行变异生成补丁程序集合.

(4) 程序验证: 程序验证器 (program validator) 使用测试用例集检验每个补丁程序. 若某个补丁程序通过了所有测试用例的测试, 表明该补丁程序对原有故障的修复符合规约, 故障修复成功并终止修复过程; 否则, 继续验证剩余的补丁程序. 如果未发现有效补丁程序并且当前语句块的所有补丁程序完成了检验, 则对下一个语句块重复步骤 (2) 和步骤 (3).

2.2 相关定义

定义 1. 基本语句块, *B*. BPEL 程序中的语句块是以 XML 标签为界定的一组语句. 基本语句块指不包含其他语句的语句块.

例如, 图 1 中第 11 行和第 12 行是一个基本语句块; 而第 1–13 行为以 XML 标签<bpel:assign> 为界定的语句块, 内嵌套<bpel:copy><bpel:from> 等, 不属于基本语句块.

定义 2. 基本语句块的类型, *BT*. 基本语句块的 XML 标签名中不含前缀的部分为基本语句块的类型.

例如, 图 1 中第 11 行和第 12 行的基本语句块的类型为“to”.

定义 3. 修复操作, *OP*. 基于故障程序生成候选补丁程序的代码修改规则.

修复操作主要基于 BPEL 程序所适用的变异算子. 由于部分变异算子所描述的代码修改规则不可逆, 无法修复由自身产生的变异体故障. 例如 BPEL 变异算子 EEU (移除表达式中的一元减法运算符) 是一个无法修复的操作, 因为无法修复因 EEU 而出现的故障, 且无其他变异算子进行有效修复. 为此, 本文对不可逆变异算子进行了扩展, 扩展的结果如表 3 所示.

表 3 扩展的 BPEL 变异算子描述

Category	Mutation operator	Description
Expression mutation	EEU	Remove or add the unary subtraction operator from an expression
	EMD	Substitute duration by 0, half of it or twice of it
	EMF	Substitute cut-off time by 0, half of it or twice of it
	EAP	Add or remove absolute values from an expression
Concurrency-independent activity mutation	ACI	Replace the value of createInstance attribute with “yes” or “no”
	AFP	Change sequential forEach activity to concurrent or concurrent to sequential
	ASF	Swap a flow activity and a sequence activity
	AIS	Replace the value of the isolated attribute with “yes” or “no”
Concurrency-related activity mutation	AWR	Swap a repeatUntil activity and a while activity

定义 4. 修复模板, *FP*. 给定类型为 *BT* 的基本语句块 *B*, 所有适用于 *BT* 的修复操作的集合 (能为该 *BT* 生成变异体的变异算子集) 称为 *BT* 的修复模板, 记为 *FP*→*BT*. 对任意 *BT*, 有且仅有一个用于指导修复过程的 *FP*.

定义 5. 修复操作集, S_{OP}. 给定一个语句块, 适用该语句块的所有 *OP* 的集合称为修复操作集, 记为 S_{OP}. 对于一个给定的语句块, 其 S_{OP} 由自身与内嵌的所有语句块类型对应的 *FP* 组成.

分析图 1 中语句块的树形结构, 可确定 *literal* 语句块的修复操作集为 S_{OP}=*FP*(*literal*), *from* 语句块的修复操作集为 S_{OP}=*FP*(*from*)∪*FP*(*literal*), 以此类推, *copy* 语句块的修复操作集为 S_{OP}=*FP*(*copy*)∪*FP*(*from*)∪*FP*(*literal*)∪*FP*(*to*). 分析每种基本语句块的类型 *BT* 适用的修复操作并构造修复模板 *FP*, 在确定故障语句块后, 根据故障语句块的修复操作集 S_{OP} 调用相应修复模板.

2.3 故障修复算法

本节介绍 BPELRepair 的故障修复算法. 算法的输入包括存在故障 BPEL 程序 *P*、测试用例集 *T*、测试用例预期输出 *E* 和修复模板集 S_{FP}. 如果修复成功, 输出为修复成功后的程序 *R*; 否则, 输出为空集. 具体步骤如算法 1.

算法 1. BPELRepair 算法.

输入: $P=\{b_1, b_2,\ldots, b_m\}$, $T=\{t_1, t_2,\ldots, t_n\}$, $E=\{e_1, e_2,\ldots, e_n\}$, S_{FP};
输出: *R*.

1. **Initialize** *ER* = *EI* = *SUS* = *BL* = *TR* = *R* = *blockTypeSet* = ∅
2. Send *T* to execute *P* and get results *ER*, *P*
3. **for** each b_i in *P* **do**
4. $sus(b_i) \leftarrow formula(Method, P, ER)$
5. **end for**
6. *BL* ← *Rank*(*SUS*) decrease
7. **for** each b_i in *BL* **do**
8. *OperatorSet* ← ∅
9. $blockTypeSet \leftarrow GetAllBlockType(b_i)$

```
10.  for bt_i in blockTypeSet do
11.    operatorSet ← GetOperatorFromPattern(bt_i, S_FP)
12.    Add operatorSet → OperatorSet
13.  end for
14.  CP ← ∅
15.  for each cb in blockTypeSet do
16.    Create candidate program with (OperatorSet, P, cb) to get TCP
17.    Add TCP → CP
18.  end for
19.  for each p in CP do
20.    TR ← Execute Program(p, T)
21.    if ( TR != E ) then
22.      break check this candidate program
23.    else then
24.      R ← p
25.      break Execute other CP
26.    end if
27.  end for
28.  if (R != ∅ ) then
29.    break Check other block
30.  end if
31. end for
32. return R
```

(1) 第 1 行: 初始化原始故障程序的测试结果集 *ER*、所有块的怀疑度集 *SUS*、有序语句块列表 *BL*、临时候选程序的执行结果 *TR*、修复结果集合 *R* 为空.

(2) 第 2 行: 对原始故障程序执行测试用例集, 记录测试用例集的执行结果 *ER*, 并记录执行轨迹 *P*.

(3) 第 3–6 行: 利用原始故障程序的执行结果与执行轨迹, 选择怀疑度计算公式计算每个语句块的怀疑度. 依据怀疑度对故障语句块降序排序.

(4) 第 7–13 行: 初始化 *OperatorSet* 集合为空; 得到语句块 b_i 下的所有基本语句块, 并将这些语句块的类型加入集合 *blockTypeSet* 中; 从 S_{FP} 中找到所有匹配的修复模板, 将模板内的修复操作加入到 *OperatorSet* 中.

(5) 第 14–18 行: 利用步骤 (4) 中 *OperatorSet* 记录的修复操作尝试对这些语句块进行修复, 生成一组经过修复的程序, 称为临时候选程序集 *TCP*; 将 *TCP* 中的候选程序加入到补丁程序集 *CP* 中.

(6) 第 19–27 行: 使用测试用例测试 *CP* 中的每一个补丁程序 *p*; 若某个补丁程序的测试结果不等于预期输出, 表明该补丁程序中的修复是无效的, 结束当前补丁程序的验证; 若某个补丁程序的测试结果完全符合预期, 表明该补丁程序中的修复是有效的, 停止验证其他补丁程序, 结果集 *R* 等于 *p*.

(7) 第 28–32 行: 判断程序修复是否成功; 成功则结束故障修复过程; 否则, 继续修复剩余语句块. 最后返回结果 *R*.

设测试用例集规模为 *L*, 原始故障程序含有 *W* 个语句块, 候选补丁程序集个数为 *N*, 计算每个语句块的怀疑度并降序排列时间复杂度为 O(*W*), 生成并验证补丁程序集时间复杂度为 O(*N*×*L*), 总时间复杂度为 O(*W*+*N*×*L*).

2.4 故障修复优化策略

为了进一步提高 BPELRepair 的故障修复效率, 从补丁生成、测试用例选择以及终止条件设定 3 个角度出发,

提出如下故障修复的优化策略.

2.4.1 补丁生成优化策略

● 模板匹配优化策略 (TMS): 目前已定义了 34 种适用于 BPEL 程序的变异算子, 而故障语句块可能只适用于其中几种变异算子. 为了减少无效的搜索匹配过程, 将 *BT* 和变异算子组合得到修复模板 *FP*. 在补丁生成时, 应用该策略后只选取 *BT* 所适用的部分变异算子, 从而提高搜索匹配的效率.

● 修复操作子集优化策略 (OPSS): 我们的前期研究工作[19]发现, BPEL 程序的变异算子存在包含关系, 即 AIE⊆AEL、CFA⊆AEL、ASI⊆ASF、CCO⊆CDC、CDE⊆CDC、EIN⊆ERR, 每一组中的变异算子能够产生相同或等价的变异体. 基于上述包含关系, 将变异算子由原先的 34 种缩减为 28 种. 通过减少修复操作 *OP* 的数量, 可以有效减少变异体的数量 (即减少补丁程序的数量), 从而缩减程序验证过程的开销.

● 历史修复操作优化策略 (HOPS): 为了更快地找到有效的修复, 给予由历史修复操作 *OP* 变异生成的补丁程序更高的优先级. Le 等人[20]认为历史故障信息能够指导后续的故障修复工作, 即假设曾经出现过故障类型可能还会发生并且概率大于少见故障类型. 通过保存历史修复报告, 给予由这些历史 *OP* 生成的补丁程序更高的优先级, 将这些补丁程序排在候选补丁程序集合的头部, 程序验证时优先检查这些补丁程序.

2.4.2 测试用例选择优化策略

验证补丁程序时, 需要执行测试用例以验证补丁程序中的修复是否有效. 当补丁程序无法通过某一个测试用例时, 也即修复无法满足程序规约, 终止该补丁程序的检验. 在此过程中, 通过优先使用高故障检测率的测试用例, 能够更快地检测出无效的补丁程序, 从而减小测试用例的执行次数并提升故障修复的效率. 相应地, 本文提出以下 3 种测试用例选择优化策略.

● 故障定位阶段错误测试用例优先策略 (DFTS): 结合故障定位阶段测试用例的测试结果, 给予检出故障的测试用例更高的执行优先级. 其核心思想是: 故障定位阶段能够检出故障的测试用例, 在补丁程序验证阶段通常也具有较大的揭示无效修复的能力.

● 覆盖故障语句块优先策略 (CSS): 结合故障定位阶段测试用例的执行覆盖情况, 给予覆盖故障语句块的测试用例更高的执行优先级. 如果在故障定位阶段某个测试用例覆盖执行了故障语句块, 那么在验证补丁程序阶段该测试用例通常也可以覆盖被修复的语句块, 因而可以快速地验证故障修复是否有效. 不同测试用例覆盖的语句块存在差异, 并且故障语句块是根据怀疑度列表的检查进度不断变化的, 因此测试用例的优先级随着修复过程发生变化.

● 验证阶段错误测试用例优先策略 (VFTS): 结合程序验证阶段测试用例的执行情况, 给予可以揭示无效修复的测试用例更高的优先级. 当执行完一次程序验证后, 如果补丁程序无法通过某个测试用例, 则表明该测试用例揭示了无效的修复; 那么, 该测试用例在未来也有较大可能揭示其他无效的修复.

2.4.3 终止条件设定策略

在故障修复过程中, 如果已经检查过真正错误的语句块之后仍未生成有效的修复程序, 那么后续的修复过程无法产生正确修复程序. 因此, 应当及时终止修复验证过程, 从而减少故障修复的开销.

定义 6. 故障修复终止位置. 终止位置指存在故障的语句块在怀疑度列表中的位置.

一般说来, 怀疑度列表中终止位置之后的语句块通常无法修复故障. 相应地, 提出以下终止条件优化策略.

● 最大语句块百分比终止策略 (SCTS): 通过设定要检查修复的语句块的最大百分比提前终止故障修复, 最大语句块百分比对应语句块的位置为故障修复终止位置.

2.5 方法示例

本节采用一个 BPEL 程序示例本文提出的故障修复技术 BPELRepair. SupplyCustomer 程序是一个商家订单管理系统. 图 3 示意了 BPEL 流程, 根据用户输入的商品信息和地址选择不同的处理方式, 当地址合法并且商品有库存时进行扣款业务和商品邮递业务. 其中扣款服务和商品邮寄服务并发执行, 最终的结果通过“replyOutput”活动返回.

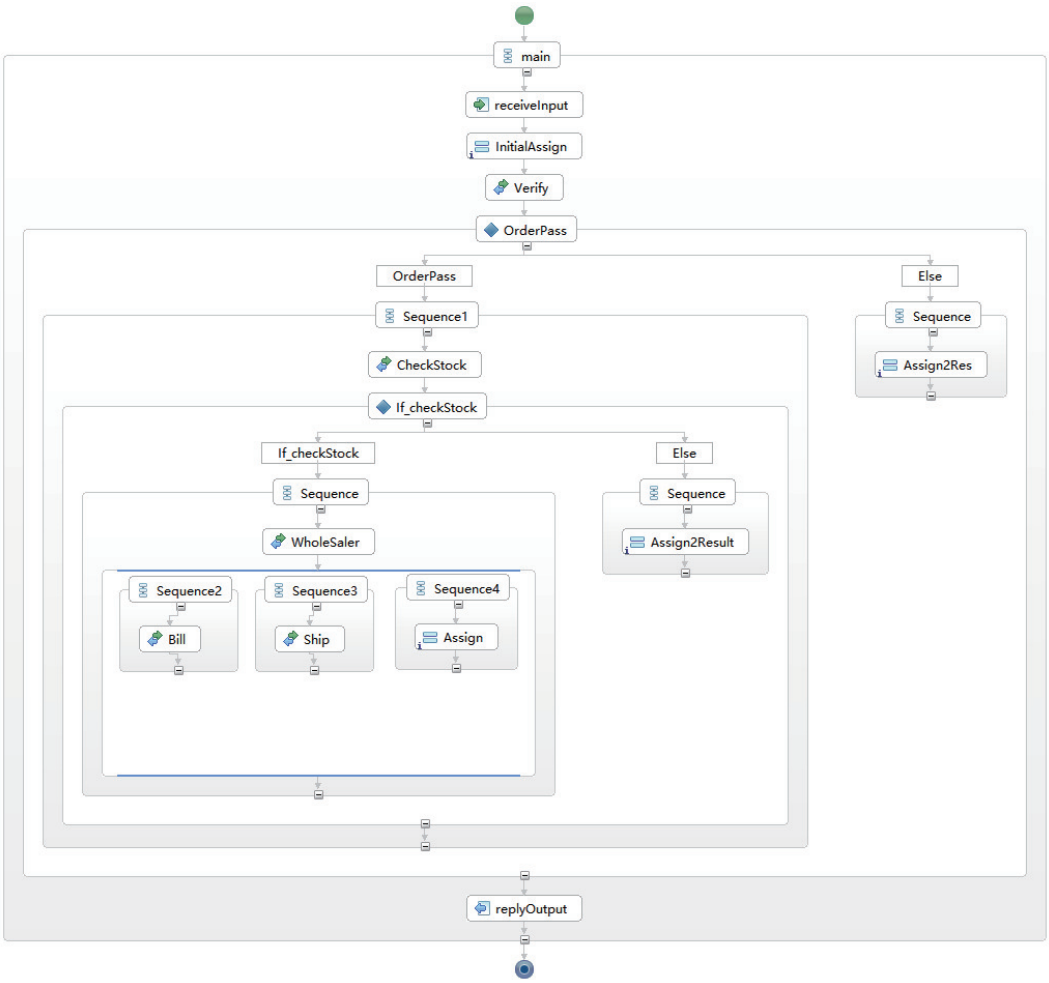

图 3 SupplyCustomer 的 BPEL 流程图

假设在 SupplyCustomer 程序实例中存在一个 ERR 类型的故障, 即“If_checkStock”语句块的条件表达式为“$checkStockResponse.parameters/ns4:checkStockReturn != ‘yes’”, 而正确程序应为“$checkStockResponse.parameters/ns4:checkStockReturn = ‘yes’”. 本文提出的 BPELRepair 工作过程如下.

步骤 1. 对待修复的故障程序 SupplyCustomer 执行配套的测试用例集, 获得每个测试用例的执行轨迹信息以及实际输出结果. 将实际运行结果与预期输出进行比对, 确定测试用例是否执行成功.

步骤 2. 选择故障定位算法, 依据步骤一的信息, 对程序中的所有语句块进行怀疑度计算并排序. 本例使用 Ochiai 公式得到怀疑度列表中语句块顺序为: If_checkStock→CheckStock→Assign2Result→InitialAssign→OrderPass→Bill→WholeSaler→Assign→Ship→Verify→Flow→Assign2Res.

步骤 3. 顺序遍历步骤 2 得到的语句块列表, 首先选择“If_checkStock”语句块. 该语句块嵌套多层语句块, 包含的基本语句块类型有 if、condition、sequence、else.

步骤 4. 从修复模板集合中, 找到步骤 3 语句块类型集合所对应的语句块模板并统计修复操作集. 涉及的修复操作包括 AIE, CFA, EAA, EEU, ERR, ELL, ECC, ECN, EIN, CDE, CCO, CDC, ASF, ASI.

步骤 5. 递归遍历“If_checkStock”语句块中所有基本语句块, 应用步骤 4 中获得的修复操作集合, 变异生成关于该语句块的所有补丁程序.

步骤 6. 逐一执行步骤 5 中生成的补丁程序, 执行该程序实例所有的测试用例, 直到存在补丁程序通过了所有的测试用例. 本例中, ERR 修复操作将条件表达式中“!=”修改为“=”从而得到了正确的程序 p. 返回修复成功的程序 p.

3 实验评估

通过经验研究评估 BPELRepair 及优化策略的有效性.

3.1 实验问题与对象

实验致力于回答以下 3 个研究问题.

(1) RQ1: BPELRepair 的故障修复有效性如何?

采用 6 个 BPEL 程序的多个故障版本, 通过故障修复率评估 BPELRepair 的有效性.

(2) RQ2: 所提的优化策略能否有效降低 BPELRepair 的故障修复开销?

针对不同的优化策略, 采用正交试验方式比较应用优化策略前后的故障修复的计算开销, 通过变异算子搜索匹配优化率、补丁程序验证优化率、测试用例执行优化率等指标评估各项优化策略的有效性.

(3) RQ3: BPELRepair 成功修复故障的代价如何?

通过分析故障修复成功时所需检查的语句块的占比, 评估 BPELRepair 的故障修复代价.

为了评估 BPELRepair 及其优化策略的有效性, 本文使用 6 个不同应用领域的 BPEL 程序作为实验对象. 表 4 总结了每个 BPEL 程序的相关信息, 包括代码行数 (lines of code)、语句块数 (blocks of code)、测试用例数 (number of test cases)、故障数量 (faults). 这些程序覆盖了伙伴链接、变量、错误处理等常见的 BPEL 组件, 具有较好的代表性. 实验程序介绍如下.

表 4 程序实例详细信息

Subject program	Lines of code	Blocks of code	Number of test cases	Faults
CarEstimate	300	9	31	23
LoanApproval	256	8	28	64
QuoteProcess	482	21	19	78
SmartShelf	705	31	144	78
SupplyCustomer	330	13	16	50
TravelAgency	407	23	69	110

● CarEstimate 是一个汽车修复评估系统, 共涉及 5 个 Web 服务. 用户提出评估请求, 输入车辆的基本信息后, 首先进行初步评估, 然后根据车辆的外观、内部以及发动机使用情况对车辆进行详细评估, 最后汇总评估结果并将其返回给用户.

● LoanApproval 是一个贷款审批系统, 共涉及 3 个 Web 服务. 输入个人信息和贷款数目后, 根据个人情况查询是否支持贷款并查询个人最大贷款金额数. 如果贷款数在允许范围内, 结果返回贷款成功信息; 否则, 返回贷款失败信息.

● QuoteProcess 是一个报价系统, 共涉及 8 个 Web 服务. 输入个人信息以及报价后, 根据该用户是否为会员选择不同的处理流程. 如果用户不是会员, 选择一般的报价评估系统; 否则, 对内部和外部做精准报价评估.

● SmartShelf 是一个智能货架管理系统, 共涉及 13 个 Web 服务. 用户输入商品的名称和数量后, 查询该产品信息, 返回当前的库存量和日期信息. 接下来, 读取相关的货架信息以及是否过期状态. 如果库存正常, 则直接返回; 如果没有库存, 则需要通知管理人员和运输人员上架产品, 并检测商品所在的货架位置是否正确. 此外, 同步检查商品保质期状态, 过期产品需及时撤出货架. 最后返回各个处理流程的执行信息.

● SupplyCustomer 是一个商家订单管理系统, 共涉及 5 个 Web 服务. 用户输入商品信息和地址并选择不同的处理方式, 当地址合法并且商品有库存时, 系统进行扣款业务和商品邮递业务. 其中, 扣款服务和商品邮寄服务并

发执行.

● TravelAgency 是一个旅行预订系统, 共涉及 8 个 Web 服务. 用户输入个人信息以及出行信息后, 选择合适的旅行社, 并且预订旅馆以及机票, 最后从会员账户扣除整个旅行过程的费用. 最后, 返回旅行预订各个流程是否成功的信息.

3.2 度量指标

介绍评估本文所提故障修复技术以及优化策略的有效性的度量指标. 采用故障修复的成功率度量 BPELRepair 的有效性; 从修复过程变异算子搜索匹配次数、验证的候选补丁数量和测试用例的执行次数等方面, 评估故障修复的计算开销. 下面介绍每个度量指标的含义及计算方法.

● 故障修复成功率 (*PF*): BPELRepair 应用于待测 BPEL 程序 p 时, 成功修复的故障与全部故障的百分比, 计算公式如公式 (1) 所示:

$$PF(p)=\frac{N_f}{N_{\text{all}}}\times 100\% \tag{1}$$

其中, N_f 表示成功修复的故障数, N_{all} 表示所有的故障数. *PF* 越大表示故障修复技术越有效.

● 变异算子搜索匹配优化率 ($OC^x_{\#Match}$): 使用某个优化策略 x 后减少的搜索匹配次数百分比, 计算公式如公式 (2) 所示:

$$OC^x_{\#Match}=\frac{\#Match_{\text{Original}}-\#Match_x}{\#Match_{\text{Original}}}\times 100\% \tag{2}$$

其中, $\#Match_{\text{Original}}$ 表示原始 BPELRepair 的变异算子搜索匹配次数, $\#Match_x$ 表示使用某个优化策略 x 后 BPELRepair 的变异算子搜索匹配次数. $OC^x_{\#Match}$ 越大表示优化后变异算子搜索匹配次数越小, 该策略越有效.

● 补丁程序验证优化率 ($OC^x_{\#Patch}$): 使用某个优化策略 x 后减少的补丁程序验证数量百分比, 计算公式如公式 (3) 所示:

$$OC^x_{\#Patch}=\frac{\#Patch_{\text{Original}}-\#Patch_x}{\#Patch_{\text{Original}}}\times 100\% \tag{3}$$

其中, $\#Patch_{\text{Original}}$ 表示原始 BPELRepair 的候选补丁数量, $\#Patch_x$ 表示使用某个优化策略 x 后 BPELRepair 验证的候选补丁数量, $OC^x_{\#Patch}$ 越大表示优化后验证的补丁程序数量越小, 该策略越有效.

● 测试用例执行优化率 ($OC^x_{\#Validation}$): 评估测试用例选择策略 x 的优化效果, 计算公式如公式 (4) 所示:

$$OC^x_{\#Validation}=\frac{\#Validation_{\text{random}}-\#Validation_x}{\#Validation_{\text{random}}}\times 100\% \tag{4}$$

其中, $\#Validation_{\text{random}}$ 表示 BPELRepair 随机选择测试用例时的测试用例执行次数, $\#Validation_x$ 表示 BPELRepair 使用了某个测试用例选择策略 x 后的测试用例执行次数. $OC^x_{\#Validation}$ 越大表示应用测试用例选择策略 x 后的测试用例执行开销越小, 优化效果越好; 当值小于 0 时, 该优化策略 x 不如随机选择测试用例策略.

3.3 实验设计

现有 BPEL 程序故障修复的主要方法有: 1) 使用正确 BPEL 程序代替存在故障的 BPEL 程序[9]; 2) 基于 BPEL 语言异常处理机制设计特定的操作进行故障修复[10,11]. 这两类方法不属于自动化的程序修复范畴, 无法作为基线技术进行比较. 本文方法中的故障定位部分采用了课题组前期研究工作成果[8], 故障定位的有效性已与同类方法进行对比, 而本文实验部分侧重于评估修复技术与优化策略的有效性.

BPEL 程序相比传统应用程序具有较大差异, 其部署执行开销都要大于传统应用程序. 本文设置 25 组不同策略组合的对比实验, 每组实验重复执行 30 次. 为了缩减计算开销, 首先生成所有故障版本程序的所有候选补丁程序, 然后对所有的候选补丁程序执行所有的测试用例集, 从而获得每一个测试用例对每一个候选补丁程序的通过情况. 进行修复实验时, 只需要查找对应程序与对应测试用例的执行结果, 避免多次执行同一个补丁程序带来的冗余开销.

表 5 列出了本文提出的多种面向补丁生成以及测试用例选择方面的优化策略, “Yes”或“No”表示实验过程是

否选用该策略. 测试用例选择策略 (1) 和策略 (2) 可以同时使用, 因为故障定位阶段未通过的测试用例有可能执行覆盖了故障语句块, 这些测试用例相比其他测试用例可能具有更高的故障检测能力. 相应地, 测试用例选择策略包括 DFTS、CSS、VFTS、随机选择测试用例策略 (random)、同时选用 DFTS 和 CSS 策略 (DFTS & CSS). 本文使用 4 种故障定位策略, 分别是 Tarantula (T)、DStar (D)、CBI (C) 和 Ochiai (O).

表 5　实验策略选项表

Strategy	Options
TMS	Yes/No
OPSS	Yes/No
HOPS	Yes/No
Test case strategy	Random/DFTS/CSS/DFTS & CSS/VFTS
Fault localization strategy	T/D/C/O

特别地, HOPS 需要历史修复报告作为参数输入, 本文使用留一法验证该策略的有效性. 首先选择 CarEstimate 之外的其他 5 个程序实例作为训练集得到历史修复操作, CarEstimate 作为测试集进行修复. 类似地, 其他 5 个程序实例依据此方式进行训练与测试过程. 实验评估所采用的修复配置为 TMS、OPSS、DFTS 与 DStar 故障定位技术.

由于不同策略的参数组合数导致实验规模庞大, 本文选用正交试验设计来缩减实验组数. 选择 L25(5^6) 正交表, 共需执行 25 组实验, 设计的正交实验方案见表 6. 为了减少实验中随机性的影响, 每组实验重复执行 30 次, 计算相应度量指标的平均值.

表 6　正交实验设计方案

No.	TMS	OPSS	Test case strategy	Fault localization strategy
1	Yes	Yes	DFTS	T
2	Yes	No	CSS	D
3	Yes	No	DFTS & CSS	C
4	Yes	No	VFTS	O
5	Yes	No	random	O
6	No	Yes	CSS	C
7	No	No	DFTS & CSS	O
8	No	No	VFTS	O
9	No	No	random	T
10	No	No	DFTS	D
11	No	Yes	DFTS & CSS	O
12	No	No	VFTS	T
13	No	No	random	D
14	No	No	DFTS	C
15	No	No	CSS	O
16	No	Yes	VFTS	D
17	No	No	random	C
18	No	No	DFTS	O
19	No	No	CSS	O
20	No	No	DFTS & CSS	T
21	No	Yes	random	O
22	No	No	DFTS	O
23	No	No	CSS	T
24	No	No	DFTS & CSS	D
25	No	No	VFTS	C

3.4 实验结果与分析

从修复成功率和计算开销两方面来评估 BPELRepair 及其优化策略的有效性.

3.4.1 BPELRepair 的有效性 (RQ1)

6 个实验程序的故障修复评估结果见表 7. 其中, faults 栏表示总的故障数量, faults successfully repaired 栏表示成功修复的故障数量, *PF* 栏表示故障修复的成功率.

表 7 故障修复评估结果

Subject program	Faults	Faults successfully repaired	*PF* (%)
CarEstimate	23	6	26.09
LoanApproval	64	26	40.63
QuoteProcess	78	50	64.10
SmartShelf	78	38	48.72
SupplyCustomer	50	31	62.00
TravelAgency	110	62	56.36
Total	403	213	52.85

由表 7 可知, BPELRepair 在 QuoteProcess 程序上的效果最好, 故障修复成功率可达 64.1%; 在 CarEstimate 上修复的效果最差, 故障修复成功率为 26.09%; BPELRepair 的平均故障修复成功率为 52.85%. 上述结果表明, BPELRepair 可以修复 50% 以上的 BPEL 程序故障.

进一步分析故障修复较低的原因时发现: 相比变异分析方式生成的故障而言, 人工植入方式生成的故障难以采用现有变异算子进行修复. 由于 CarEstimate 与 LoanApproval 中存在较多的人工植入故障类型, 因此故障修复成功率相对较低. 举例说来, 图 4 示意了一个程序及其故障版本. 故障程序图 4(b) 在第 4 行与第 15 行出现了两种错误且都无法采用现有变异算子修复: 1) nsl 的 name 赋值错误 (“cookie”→“tns:name”): 当修复程序定位到该故障时, 将使用变异算子 ISV (相同类型的变量标识符替换) 修复, 根据上下文找出相同类型的变量标识符并替换生成变异体, 但正确程序实际上使用了上下文没有出现过的新字符串, 因此修复失败; 2) nsl 的 amount 赋值错误 (“6060” → “6000”): 当修复程序定位到该故障时, 将使用 ECN (增加或减小一个数字常量的值, 添加或移除一个数字)、EAP (用表达式的绝对值替代该表达式)、EAN (用表达式的绝对值的负值替代该表达式) 等变异算子修复, 然而这些变异算子并不能将数值 6 000 成功修复为 6 060.

```
1  <bpel:copy>
2    <bpel:from part="payload" variable="input">
3      <bpel:query >
4        <![CDATA[“cookie”]]>
5      </bpel:query>
6    </bpel:from>
7      <bpel:to part="parameters" variable="initialRequest">
8        <bpel:query>
9          <![CDATA[ns1:name]]>
10       </bpel:query>
11     </bpel:to>
12   </bpel:copy>
13   <bpel:copy>
14     <bpel:from>
15       <![CDATA[6060]]>
16     </bpel:from>
17     <bpel:to part="parameters" variable="initialRequest">
18       <bpel:query>
19         <![CDATA[ns1:amount]]>
20       </bpel:query>
21     </bpel:to>
22   </bpel:copy>
```

(a) 正确程序

```
1  <bpel:copy>
2    <bpel:from part="payload" variable="input">
3      <bpel:query >
4        <![CDATA[tns:name]]>
5      </bpel:query>
6    </bpel:from>
7      <bpel:to part="parameters" variable="initialRequest">
8        <bpel:query>
9          <![CDATA[ns1:name]]>
10       </bpel:query>
11     </bpel:to>
12   </bpel:copy>
13   <bpel:copy>
14     <bpel:from>
15       <![CDATA[6000]]>
16     </bpel:from>
17     <bpel:to part="parameters" variable="initialRequest">
18       <bpel:query>
19         <![CDATA[ns1:amount]]>
20       </bpel:query>
21     </bpel:to>
22   </bpel:copy>
```

(b) 故障程序

图 4 故障修复失败示例

3.4.2 故障修复优化策略的效果 (RQ2)

表 8 和表 9 总结了运用正交实验与留一法得到优化策略应用前后的实验结果均值. 其中, *#Match*、*#Patch*、*#Validation* 分别表示变异算子搜索匹配次数、验证的候选补丁数、测试用例执行数等计算开销; original、optimized 分别表示原始 BPELRepair 与优化后的 BPELRepair 的计算开销; *OC* 表示使用某个策略后 BPELRepair 的优化率, average 表示 6 个实验程序的平均优化率, 反映了优化策略的有效性. 由表 8 可知如下.

表 8 补丁生成优化策略的有效性评估结果

Computation cost	Optimization strategy	Optimization effectiveness	Car-Estimate	Loan-Approval	Quote-Process	Smart-Shelf	Supply-Customer	Travel-Agency	Average
#Match	TMS	Original	136.67	92.09	123.98	261.54	75.12	244.94	—
		Optimized	29.00	32.08	28.96	49.92	20.97	63.42	—
		$OC^{TMS}_{\#Match}$ (%)	78.78	65.16	76.61	80.91	72.09	74.11	74.61
	OPSS	Original	119.30	83.13	108.90	227.39	66.67	216.41	—
		Optimized	98.47	67.92	89.31	186.51	50.78	177.55	—
		$OC^{OPSS}_{\#Match}$ (%)	17.46	18.29	17.98	17.97	23.83	17.95	18.91
#Patch	OPSS	Original	83.64	75.57	43.96	103.74	45.91	127.45	—
		Optimized	79.47	64.52	35.81	84.18	41.92	106.36	—
		$OC^{OPSS}_{\#Patch}$ (%)	4.99	14.62	18.53	18.85	8.69	16.54	13.70
	HOPS	Original	81.50	69.53	38.34	87.28	43.51	112.86	—
		Optimized	65.35	66.34	29.38	80.29	38.18	107.69	—
		$OC^{HOPS}_{\#Patch}$ (%)	19.81	4.59	23.35	8.01	12.26	4.58	12.10
#Validation	OPSS	Original	133.30	218.58	115.03	491.37	90.42	474.07	—
		Optimized	130.37	184.87	102.46	446.69	82.62	453.66	—
		$OC^{OPSS}_{\#Validation}$ (%)	2.20	15.42	10.92	9.09	8.63	4.31	8.43
	HOPS	Original	114.07	117.06	68.64	323.78	67.81	210.37	—
		Optimized	108.10	110.79	60.56	313.13	59.18	195.06	—
		$OC^{HOPS}_{\#Validation}$ (%)	5.23	5.36	11.76	3.29	12.72	7.28	7.61

● 模板匹配优化策略 TMS 有效减少变异算子搜索匹配开销. 6 个实验程序的平均变异算子搜索匹配优化率为 74.61%; 其中 SmartShelf 程序的优化率为 80.91%, 意味着极大地减少了搜索匹配次数.

● 修复操作子集优化策略 OPSS 减少了变异算子搜索匹配次数、验证的候选补丁数与测试用例执行数, 平均优化率分别为 18.91%、13.70% 与 8.43%.

● 历史修复操作优化策略 HOPS 减少了验证的候选补丁数与测试用例执行开销. 具体说来, 平均减少 12.10% 候选补丁验证开销、7.61% 的验证候选程序测试用例执行开销. 实验结果验证了历史故障类型可能再次发生的假设合理性, 优先检查历史修复操作生成的候选程序可以更快地找到正确修复程序.

表 9 总结了所提测试用例选择优化策略的评估结果. 具体说来:

● 测试用例选择策略 DFTS 与 VFTS 可以减少测试用例执行开销: 相对于随机策略 random 而言, 平均测试用例执行优化率分别为 51.68% 和 53.21%. 上述结果表明: 在故障定位阶段能够检出 BPEL 程序故障的测试用例有较强的无效修复检测能力; 在验证阶段能够检测错误修复的测试用例有较强的检测其他故障修复的能力.

● 测试用例选择策略 CSS 的优化效果不明显: 测试用例执行开销略高于 random; 且策略 DFTS 和 CSS 合并后的优化效果略低于策略 DFTS. 进一步分析发现:

(1) 如果当前语句块是故障语句块, 那么优先选择覆盖故障语句块的测试用例一定能够覆盖当前语句块, 相比未覆盖该语句块的测试用例而言, 有更强检测错误修复的能力. 由于无法确定故障语句块的真实位置, 从而无法保证测试用例执行覆盖真实故障语句块. 如果当前语句块不是故障语句块, 那么 CSS 策略将无法保证覆盖真实的故

障语句块, 甚至无法检测错误的修复.

(2) 能检出故障的测试用例一定覆盖故障语句块, 但仍然无法确定故障语句块真实位置, 当 CSS 策略与 DFTS 策略结合时, 真正起作用的则是 DFTS 策略.

表 9 测试用例选择优化策略的#Validation 评估结果

Subject program	Random	DFTS		CSS		DFTS & CSS		VFTS	
		Optimized	$OC^{DFTS}_{\#Validation}$ (%)	Optimized	$OC^{CSS}_{\#Validation}$ (%)	Optimized	$OC^{DFTS\&CSS}_{\#Validation}$ (%)	Optimized	$OC^{VFTS}_{\#Validation}$ (%)
CarEstimate	158.43	116.71	26.33	161.65	−2.03	115.52	27.08	116.01	26.78
LoanApproval	337.82	122.38	63.77	339.33	−0.45	124.27	63.21	134.12	60.30
QuoteProcess	172.35	74.08	57.02	173.68	−0.77	74.65	56.69	81.14	52.92
SmartShelf	720.95	355.50	50.69	728.74	−1.08	359.42	50.15	266.74	63.00
SupplyCustomer	117.61	71.46	39.24	120.53	−2.48	70.99	39.64	68.60	41.67
TravelAgency	840.16	226.74	73.01	852.92	−1.52	228.59	72.79	213.77	74.56
Average	—	—	51.68	—	−1.39	—	51.59	—	53.21

3.4.3 BPELRepair 成功修复的代价 (RQ3)

图 5 展示了 BPELRepair 应用于不同实验对象时成功修复故障的代价, 即修复故障版本时, 检查的语句块数占总语句块数的百分比. 其中, 盒的中间线代表了数据的中位数; 盒的上下底分别是数据的上四分位数和下四分位数; 上下边缘则代表了该组数据的最大值和最小值; 盒内点为均值; 盒外部点为数据中的“异常值”“Total”汇总了所有实验程序的成功修复代价, 均值为 28.61%. 实验结果表明 BPELRepair 的故障修复的代价是可接受的.

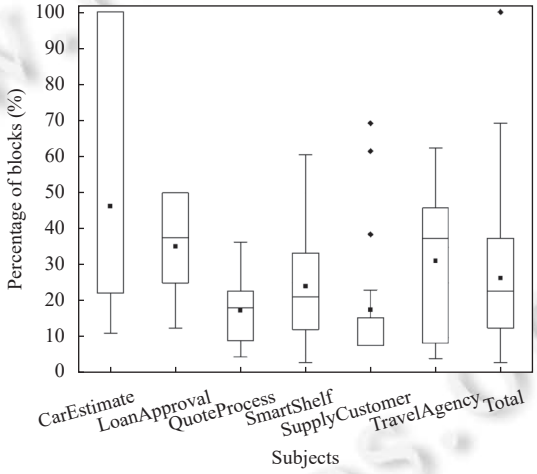

图 5 BPELRepair 的故障修复代价评估结果 (盒图)

4 相关工作

本节首先介绍基于搜索的故障修复技术相关的研究工作, 然后介绍 BPEL 程序故障修复技术与定位技术的研究工作.

4.1 故障修复技术

程序故障是指程序实际执行时产生的行为和预期行为发生了偏差. 故障修复是对程序的非预期行为进行修正的过程[21,22]. 现有的故障修复技术大致分为 3 类[23]: 基于搜索的方法、基于语义的方法和其他类型的方法. 基于搜索的故障修复技术又被称为“生成—验证”技术, 每生成一个候选补丁程序, 对其执行测试用例集以验证其是否为

“正确”的修复. 基于搜索的故障修复技术分为 3 类: 原子修改操作符技术、预定义模板技术、样例模板技术.

原子修改操作符技术 (atomic change operators) 通过对程序的单个位置执行插入、删除或替换操作从而生成补丁程序. Goues 等人[24]提出了 GenProg 算法, 通过遗传规划算法对 C 语言的抽象语法树进行操作来生成补丁程序, 执行测试用例集以判断补丁程序是否有效修复了故障. Qi 等人[25]认为 GenProg 方法所采用的遗传规划算法存在计算开销庞大的问题, 提出了一种用于搜索补丁程序的简单搜索策略; 实验结果表明简单搜索在 24 个故障中的 23 个故障版本中都能更快地找到正确的修复程序, 并且测试用例执行开销更小. Villanueva 等人[26]将新颖搜索应用于 GenProg, 避免局部最优问题, 从而获得更优的补丁程序. Debroy 等人[17]提出了基于变异分析的故障修复技术, 通过替换相同类型的原子更改操作符 (例如算术运算符、关系运算符、逻辑运算符等) 生成补丁程序并进行验证.

预定义模板技术 (pre-defined templates) 根据预定义模板中的修改操作符集合修改故障程序从而生成候选补丁程序. 与原子修改操作符技术相比, 可以同时修改一个或多个语句. 开发人员可以定义复杂的修复模板来修复一些复杂的故障甚至多故障程序. 文献 [27] 提出了阶段条件合成算法 SPR, 该算法第 1 阶段将一组通用模板应用于故障程序, 对程序进行转换; 第 2 阶段确定模板中的参数并尝试合成条件以使补丁程序通过测试用例. 将阶段式修复和条件结合能生成并有效地搜索丰富的故障修复空间.

样例模板技术 (example-based templates) 通过参考真实软件故障的修复方案设计软件故障的修改操作符, 并构造模板. Kim 等人[28]提出基于模式的自动故障修复技术 (PAR), 人工分析了 6 万个真实故障修复并总结出一套模板用于指导修复过程. 历史驱动的故障修复技术[20]通过提取多个软件项目中的历史信息来指导候选修复的生成, 利用从项目中挖掘得到的信息指导合成故障修复模板, 最后利用修复模板对故障程序进行修改获得候选解决方案. 该技术与 PAR 均基于如下假设: 依据真实故障修复案例定义的修复模板比其他模板更有效, 并且可能提高测试人员对生成的补丁的可接受性. 不同的是, 历史驱动的故障修复技术侧重软件项目中的历史信息来指导补丁程序的生成, 而 PAR 只分析现有人工编写的补丁来总结修复模式. Islam 等人[29]通过对 5 个软件系统的 4 653 个故障版本进行分析, 提出了 38 种故障修复模板以及 37 种嵌套代码结构的模板用于指导故障修复过程. 嵌套模板为故障修复模板的理解提供了一个新的维度, 也提高了故障定位和生成有效补丁的概率.

由于 BPEL 程序得故障版本较少, 样例模板技术不再适用. 本文提出的 BPELRepair 基于预定义模板技术, 在考虑 BPEL 程序特点的基础上得到高效的修复模板, 并依据修复模板中的变异算子生成 BPEL 故障程序的补丁, 若通过测试则表明相应的故障被成功修复了.

4.2 BPEL 程序的故障修复技术

针对 BPEL 程序的新特点, 人们提出了一些面向 BPEL 程序的故障修复技术. 利用 BPEL 协调参与部分 Web 服务的特点, Subramanian 等人[9]对常见的 BPEL 程序故障进行了分类, 并设计了相应的修复操作. 其中, 对于单个程序的功能性故障类型 (逻辑错误相关) 而言, 假设存在正确的 BPEL 程序版本, 修复操作采用正确版本替换故障版本. 此方法并没有讨论如何修复 BPEL 程序中的故障, 因此不属于传统意义上的故障修复范畴. Modafferi 等人[10]结合 BPEL 语言提供的事件、补偿、异常等处理机制, 以标注的方式预先定义可能出现故障时的特定操作, 包括终止、重做、回滚、警报、替换等, 开发了引擎 SH-BPEL 支持特定操作的实现. 类似地, 文献 [11] 将 SH-BPEL 作为 BPEL 程序的执行与修复引擎, 在 BPEL 程序故障出现后, 依据程序的结构约束与数据依赖定位故障, 以减少不必要的修复操作. 上述方法的修复操作需要人为设定, 因而不属于自动化的程序修复范畴.

本文所提 BPEL 程序故障修复技术主要针对功能性故障, 采用变异分析的方式生成修复模板; 结合 BPEL 程序的特点, 提出了基于变异分析的程序修复的优化策略, 支持高效的 BPEL 程序故障自动化修复.

4.3 BPEL 程序的故障定位技术

故障定位负责确定故障语句的位置以便指导后续的故障修复[21], 通过量化故障程序中的每一条语句的故障怀疑度形成降序的怀疑度列表. 一个程序所有的预期行为的集合就是程序规约[21], 程序规约可以由自然语言、形式化语言或者测试用例集表示.

近年来, 研究人员从不同角度探索了 BPEL 程序的故障定位技术. Sun 等人[6]提出一种基于语句块的 BPEL 程序故障定位框架. 该框架结合 BPEL 中语句块特点, 利用测试用例的执行信息计算每个语句块的怀疑度并提供检查语句块的启发式规则, 定位精度可达 80%, 但定位代价偏高, 效率仅达到 50%. 基于谓词切换的 BPEL 程序故障定位技术[7]通过谓词切换逐步缩小程序的故障范围, 其核心在于采用程序切片技术分析与关键谓词相关的程序片段, 该技术定位效率达 90%, 但定位精度存在一定的不足, 最高达 50%. Sun 等人[19]提出一种基于谓词切换和动态切片技术相结合的 BPEL 程序故障定位技术, 首先利用谓词切换技术找到程序的关键谓词. 然后利用后向切片算法, 搜索到起始节点, 最后输出排序后的故障语句块集合, 该方法具有较高的故障定位召回率和较低的定位代价, 但无法定位出变量声明和活动交换等故障类型. 孙昌爱等人[8]将变异测试应用到 BPEL 程序的故障定位中, 提出一种基于变异分析的 BPEL 程序故障定位技术. 首先通过变异获得故障程序所有的变异体, 对 BPEL 故障程序和变异体程序执行测试用例集以获得详细的执行信息. 统计每个语句块对应变异体集的执行信息并结合怀疑度计算公式计算语句块的怀疑度, 最后根据怀疑度的值对语句块进行降序排列. 该技术借鉴基于语句块的 BPEL 程序故障定位框架, 在提高 BPEL 程序故障定位召回率的同时, 有效降低了变异执行开销.

5 总 结

BPEL 服务组合的质量问题日益受到广泛关注. 已有工作探讨了面向 BPEL 程序的故障定位技术, 但是尚未针对 BPEL 程序故障修复开展研究工作. 本文结合 BPEL 程序的特性以及变异分析技术, 提出一种基于模板匹配的 BPEL 程序故障修复技术 BPELRepair, 通过 BPEL 程序的语句块类型信息与变异算子组合构造修复模板, 依据修复模板对故障语句块执行变异操作生成候选补丁程序, 通过执行测试用例集验证补丁程序的正确性. 从补丁程序生成、测试用例选择以及终止条件 3 个角度, 提出了多种有效的优化策略. 通过 6 个 BPEL 程序评估所提故障修复技术及优化策略的有效性. 实验结果表明: 本文所提故障修复技术能够成功修复部分故障版本; 补丁生成优化策略能够有效降低故障修复开销; 提出的两种测试用例选择策略能够减少 26%–75% 的测试用例执行开销; 提出的故障修复终止策略能够在不显著影响故障修复成功率的前提下减少不必要的语句块修复验证开销.

进一步的研究工作包括: (1) 已有的变异算子并不能覆盖所有故障类型, 未来将扩展更多 BPEL 变异算子, 增大故障修复搜索空间, 从而修复更复杂的故障类型; (2) 本文故障定位与修复阶段都使用变异分析, 但实现的工具只利用了定位阶段测试用例信息, 我们将综合两个阶段变异算子的使用信息, 进一步提高定位修复效率.

References:

[1] Goeb A, Lochmann K. A software quality model for SOA. In: Proc. of the 8th Int'l Workshop on Software Quality. Szeged: ACM, 2011. 18–25. [doi: 10.1145/2024587.2024593]

[2] IBM. What is SOA? 2021. https://www.ibm.com/cloud/learn/soa

[3] Wright G. BPEL (business process execution language). 2022. https://www.techtarget.com/searchapparchitecture/definition/BPEL-Business-Process-Execution-Language

[4] Wang XH, Zhao FY. Research summary for the key technology of software fault localization. Software Guide, 2017, 16(7): 205–209 (in Chinese with English abstract). [doi: 10.11907/rjdk.171181]

[5] Boubeta-Puig J, Medina-Bulo I, García-Domínguez A. Analogies and differences between mutation operators for WS-BPEL 2.0 and other languages. In: Proc. of the 4th Int'l Conf. on Software Testing, Verification and Validation Workshops. Berlin: IEEE, 2011. 398–407. [doi: 10.1109/ICSTW.2011.52]

[6] Sun CA, Zhai YM, Shang Y, Zhang ZY. BPELDebugger: An effective BPEL-specific fault localization framework. Information and Software Technology, 2013, 55(12): 2140–2153. [doi: 10.1016/j.infsof.2013.07.009]

[7] Sun CA, Ran YF, Zheng CY, Liu H, Towey D, Zhang XY. Fault localisation for WS-BPEL programs based on predicate switching and program slicing. Journal of Systems and Software, 2018, 135: 191–204. [doi: 10.1016/j.jss.2017.10.030]

[8] Sun CA, Zhang SF, Zhu WZ. Mutation based fault localization technique for BPEL programs. Computer Science, 2021, 48(1): 301–307 (in Chinese with English abstract). [doi: 10.11896/jsjkx.200900051]

[9] Subramanian S, Thiran P, Narendra NC, Mostefaoui GK, Maamar Z. On the enhancement of BPEL engines for self-healing composite Web services. In: Proc. of the 2008 Int'l Symp. on Applications and the Internet. Turku: IEEE, 2008. 33–39. [doi: 10.1109/SAINT.2008.

12]

[10] Modafferi S, Mussi E, Pernici B. SH-BPEL: A self-healing plug-in for Ws-BPEL engines. In: Proc. of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC 2006). Melbourne: ACM, 2006. 48–53. [doi: 10.1145/1169091.1169099]

[11] Friedrich G, Fugini MG, Mussi E, Pernici B, Tagni G. Exception handling for repair in service-based processes. IEEE Trans. on Software Engineering, 2010, 36(2): 198–215. [doi: 10.1109/TSE.2010.8]

[12] Papadakis M, Le Traon Y. Using mutants to locate "unknown" faults. In: Proc. of the 5th Int'l Conf. on Software Testing, Verification and Validation. Montreal: IEEE, 2012. 691–700. [doi: 10.1109/ICST.2012.159]

[13] DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. Computer, 1978, 11(4): 34–41. [doi: 10.1109/C-M.1978.218136]

[14] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]

[15] Papadakis M, Le Traon Y. Effective fault localization via mutation analysis: A selective mutation approach. In: Proc. of the 29th Annual ACM Symp. on Applied Computing. Gyeongju: ACM, 2014. 1293–1300. [doi: 10.1145/2554850.2554978]

[16] Estero-Botaro A, Palomo-Lozano F, Medina-Bulo I, Domínguez-Jiménez JJ, García-Domínguez A. Quality metrics for mutation testing with applications to WS-BPEL compositions. Software Testing, Verification and Reliability, 2015, 25(5–7): 536–571. [doi: 10.1002/stvr.1528]

[17] Debroy V, Wong WE. Using mutation to automatically suggest fixes for faulty programs. In: Proc. of the 3rd Int'l Conf. on Software Testing, Verification and Validation. Paris: IEEE, 2010. 65–74. [doi: 10.1109/ICST.2010.66]

[18] Zheng CY. A predicate switching based approach to locating faults of BPEL programs and supporting tool [MS. Thesis]. Beijing: University of Science and Technology Beijing, 2015 (in Chinese with English abstract).

[19] Sun CA, Wang Z, Pan L. Optimized mutation testing techniques for WS-BPEL programs. Journal of Computer Research and Development, 2019, 56(4): 895–905 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2019.20180037]

[20] Le XBD, Lo D, Le Goues C. History driven program repair. In: Proc. of the 23rd Int'l Conf. on Software Analysis, Evolution, and Reengineering. Osaka: IEEE, 2016. 213–224. [doi: 10.1109/SANER.2016.76]

[21] Li B, He YP, Ma HT. Automatic program repair: Key problems and technologies. Ruan Jian Xue Bao/Journal of Software, 2019, 30(2): 244–265 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5657.htm [doi: 10.13328/j.cnki.jos.005657]

[22] Gazzola L, Micucci D, Mariani L. Automatic software repair: A survey. IEEE Trans. on Software Engineering, 2019, 45(1): 34–67. [doi: 10.1109/TSE.2017.2755013]

[23] Wang Z, Gao J, Chen X, Fu HJ, Fan XY. Automatic program repair techniques: A survey. Chinese Journal of Computers, 2018, 41(3): 588–610 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00588]

[24] Goues CL, Nguyen T, Forrest S, Weimer W. GenProg: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012, 38(1): 54–72. [doi: 10.1109/TSE.2011.104]

[25] Qi YH, Mao XG, Lei Y, Dai ZY, Wang CS. The strength of random search on automated program repair. In: Proc. of the 36th Int'l Conf. on Software Engineering. Hyderabad: ACM, 2014. 254–265. [doi: 10.1145/2568225.2568254]

[26] Villanueva OM, Trujillo L, Hernandez DE. Novelty search for automatic bug repair. In: Proc. of the 2020 Genetic and Evolutionary Computation Conf. Cancún: ACM, 2020. 1021–1028. [doi: 10.1145/3377930.3389845]

[27] Long F, Rinard M. Staged program repair with condition synthesis. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 166–178. [doi: 10.1145/2786805.2786811]

[28] Kim D, Nam J, Song J, Kim S. Automatic patch generation learned from human-written patches. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 802–811. [doi: 10.1109/ICSE.2013.6606626]

[29] Islam MR, Zibran MF. How bugs are fixed: Exposing bug-fix patterns with edits and nesting levels. In: Proc. of the 35th Annual ACM Symp. on Applied Computing. Brno: ACM, 2020. 1523–1531. [doi: 10.1145/3341105.3373880]

附中文参考文献:

[4] 黄小红, 赵逢禹. 软件故障定位关键技术研究综述. 软件导刊, 2017, 16(7): 205–209. [doi: 10.11907/rjdk.171181]

[8] 孙昌爱, 张守峰, 朱维忠. 一种基于变异分析的BPEL程序故障定位技术. 计算机科学, 2021, 48(1): 301–307. [doi: 10.11896/jsjkx.200900051]

[18] 郑彩云. 基于谓词切换的BPEL程序故障定位技术与支持工具研究 [硕士学位论文]. 北京: 北京科技大学, 2015.

[19] 孙昌爱, 王真, 潘琳. 面向WS-BPEL程序的变异测试优化技术. 计算机研究与发展, 2019, 56(4): 895–905. [doi: 10.7544/issn1000-

1239.2019.20180037]
[21] 李斌, 贺也平, 马恒太. 程序自动修复: 关键问题及技术. 软件学报, 2019, 30(2): 244–265. http://www.jos.org.cn/1000-9825/5657.htm [doi: 10.13328/j.cnki.jos.005657]
[23] 王赞, 郜健, 陈翔, 傅浩杰, 樊向宇. 自动程序修复方法研究述评. 计算机学报, 2018, 41(3): 588–610. [doi: 10.11897/SP.J.1016.2018.00588]

孙昌爱(1974－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件测试, 故障定位, 服务计算.

张守峰(1997－), 男, 硕士生, 主要研究领域为软件测试.

吴思懿(1998－), 女, 硕士生, 主要研究领域为软件测试.

付安(1993－), 男, 博士生, 主要研究领域为软件测试.