文章编号:1671-6833(2021)02-0061-06

基于模板挖掘的程序自动修复方法

韩俊璇, 孙伟峰, 赵瑞莲, 王微微

(北京化工大学 信息科学与技术学院,北京 100029)

摘 要: 在程序修复模板挖掘的过程中,由于历史修复信息中存在噪声干扰,导致挖掘出的修复模板可用性不强,程序修复效果不佳。再者,利用修复模板对程序缺陷进行修复是程序自动修复的关键。从修复模板挖掘和程序自动修复两方面出发,提出了一种改进的程序自动修复方法 APRMT(automatic program repair method based on template mining)。APRMT 通过正则匹配消除历史修复信息中的噪声,提高修复模板的准确性;依据程序缺陷代码位置与类型,采用最近最相关策略搜索针对程序缺陷的修复信息,辅助修复模板对程序缺陷进行自动修复。为验证该方法的有效性,APRMT 针对 4 个开源项目进行实验,结果表明:APRMT 成功修复了 26 个缺陷,平均每个缺陷的修复时间为 3.1 min。由此可见,APRMT 能在更短时间内修复更多的程序缺陷,提升程序自动修复的效果及效率。

关键词: 自动修复; 模板; 程序; 缺陷; 挖掘

中图分类号: TP311.5 **文献标志码:** A **doi:**10.13705/j.issn.1671-6833.2021.02.009

0 引言

程序自动修复旨在利用程序相关信息,对含有缺陷的程序进行自动修复。修复生成-验证技术是近年来提出的一种通用程序自动修复方法,它首先对程序源代码进行修改,生成一组补丁,然后通过执行测试用例,验证补丁的正确性。GenProg[1]是最早提出的生成-验证工具,它使用变异来修改源程序以生成补丁,结合遗传规划算法中的适应度函数选取更优的补丁进入下一代,并通过执行测试用例验证每一代补丁的正确性。然而,其适应度函数仅与测试用例通过与否相关,易导致补丁虽能通过所有测试用例,但与源程序的语义偏差极大。由于 GenProg 的修复对象仅为 C 程序,Martinez 等[2]提出了 jGenProg 来对 GenProg 进行扩展,实现了对 Java 程序的自动修复。相较于 GenProg,Kim 等[3]提出了一种通过随机搜索算法来生成补丁的方法 RSRepair,该方法虽未采用适应度来指导补丁生成,但其修复效果与遗传规划算法基本相同。然而,此类程序自动修复技术由于缺少相关修复信息的指导,生成的补丁数量虽然较多,但仅有极少数补丁与源程序语义相同,修复效果不佳。

不同于上述方法,基于模板的程序自动修复方法可从历史修复信息中获取与缺陷相关的修改信息,如修改位置、修复目标等信息,进而指导程序的自动修复。此类方法生成的补丁与源程序语义具有较高的相似性,可提升程序自动修复的效果。近年来,研究人员提出了许多修复模板挖掘的方法来实现程序的自动修复。例如:Liu 等[4]从 StackOverflow 问答信息出发,提出一种程序修复模板挖掘的方法 SOFIX。然而,SOFIX 在挖掘修复信息时,未对修复信息进行完善的预处理,导致模板的修复效果不佳。此外,SOFIX 在搜索修复模板时,采用枚举策略完成程序自动修复,致使程序自动修复的效率不高。Le 等[5]提出一种从历史修复信息中总结修复模板的方法 HistoricalFix,但该修复模板在修复数量和粒度上存在局限性。Xuan 等[6]提出一种面向条件语句修复模板挖掘的方法 Nopol,该方法仅支持 if 条件语句的缺陷修复。

由上述分析可知,现有修复模板挖掘方法未对修复信息进行深入分析,忽略了修复信息中的部分语义,导致挖掘的修复模板不够准确。此外,

收稿日期:2020-11-22;**修订日期:**2020-12-28

基金项目:国家自然科学基金资助项目(61672085)

通信作者:王微微(1990—),女,河北沧州人,北京化工大学师资博士后,主要从事软件测试研究,E-mail:wangww@mail.buct.edu.cn。

现有研究大多关注修复模板挖掘方法,对于如何利用修复模板对程序进行修复尚未给出具体方法。

因此,本文提出了一种改进的程序自动修复方法 APRMT,包括修复模板的挖掘和基于修复模板的程序自动修复两部分。具体来说,APRMT 通过正则匹配规则消除修复信息中的无用信息,避免噪声信息的干扰,精确分析修复信息的语义,识别不完整的修改操作,并对其补全,避免因信息缺失导致的修复模板不准确问题。在此基础上,APRMT 依据缺陷代码位置及类型,采用最近最相关策略搜索当前程序缺陷的相关修复信息,解决了生成补丁与源代码语义差别大的问题,缩短了程序自动修复时间,提高了修复效率。

1 APRMT 修复模板挖掘

1.1 程序代码及修复信息的获取及预处理

APRMT 方法旨在从 StackOverflow 问答对的程序修复信息中挖掘修复模板,指导程序的自动修复[7]。因此,本节首先探讨如何从 StackOverflow 问答对中提取程序及其修复信息,以及对此类信息的预处理,为后续模板挖掘奠定基础。

StackOverflow 问答对中提供了大量关于程序修复的代码段信息,即代码对<bug,fixed>,其中 bug 表示含有缺陷的代码段,fixed 表示其缺陷修复后的代码段。但是在 StackOverflow 中,一个 bug 代码段可能存在多种 fixed 代码段作为其修复结果。为了尽可能保证两段代码语义的一致性,在选择代码对时,应保留对 bug 代码段修改较小的 fixed 代码段。

为提高后续模板挖掘的有效性,采用 APRMT 方法对代码对进行预处理,首先通过正则表达式消除问答代码对中的无用信息,提高代码对相似度比较的准确性;然后,采用 APRMT 计算代码对中两段代码的相似程度,并尽可能选取相似度较高的代码对作为后续模板挖掘的基础。在进行代码相似度计算时,TF-IDF 算法可为代码中较为重要的单词赋予较高的权重,相比纯文本匹配相似度,基于单词权重计算出的代码相似度更高。因此,本文选用 TF-IDF 算法计算代码之间的相似程度。

1.2 修复模板挖掘

修复信息的获取及预处理是修复模板挖掘的基础。因此,本节基于筛选出的代码对<bug,fixed>,提取从 bug 到 fixed 的修改序列,并采用差异节点追踪算法完善修复信息的语义。在此基础上,对同构的修改序列进行整理归纳得出修复模板。

代码对<bug,fixed>的修改序列提取采用抽象语法树差异分析提取工具 Gumtree[8] 实现,但该工具不能有效处理差异点是同一父节点的情况。例如,若从 bug 代码段到 fixed 代码段的修改序列是在同一个父节点下进行的移动操作,Gumtree 无法获取到该移动操作;若修改序列是对一整棵子树的移动、删除或插入,Gumtree 会将整个动作分解为对各个节点的操作,这样会增加修改序列的复杂性。

为提高模板挖掘的准确性,APRMT 在提取修改序列的过程中,通过对代码对<bug,fixed>的差异节点进行追踪,将代码对中涉及的差异节点进行完整输出,并对连续关联的操作进行整合,有效避免了上述 Gumtree 的缺点。

在获取到具有完整修复语义信息的修改序列后,对具有相同构造的修改序列进行归纳整理,总结出修复模板。APRMT 最终确定了 11 个修复模板,各模板的名称、样式及模板操作元素如表 1 所示。

1.3 模板操作元素

修复模板是对抽象语法树的修复描述,包括节点的更新(update)、删除(delete)、移动(move)以及插入(insert)等操作。模板操作元素对应于程序缺陷的缺陷类型与目标修复类型,即以上 4 种操作的操作对象及其结果的类型。例如表 1 中的修复模板变量替换,模板样式中 var1 和 var2 为该修复模板的模板操作元素,其中 var1 表示缺陷模板操作元素,var2 表示目标模板操作元素。本文挖掘的修复模板中这 2 个模板操作元素的类型是一致的,所以在表 1 第 3 列均用同一类型表示。从表 1 可以看出,修复模板的模板操作元素类型包括变量、语句、方法、变量类型以及二元操作符等。

APRMT 对模板操作元素的使用场景涉及以下 2 种:①在修复程序时通过计算缺陷代码类型与缺陷模板操作元素的相似度生成模板池;②在运用修复模板修复程序缺陷时,利用目标模板操作元素从缺陷程序中搜索有效的修复信息,完成程序的自动修复。

2 APRMT 程序自动修复

在挖掘的 11 个修复模板的基础上,本文提出

表 1　挖掘的修复模板

Table 1　Mined repair template

模板名称	模板样式	模板操作元素
变量替换	update (var1) at (pos) to (var2)	var
语句移除	delete (statement) at (pos)	statement
方法调用替换	update (method1) at (pos) to (method2)	method
参数添加	insert (var) at (pos) in (argument)	var
参数移除	delete (var) at (pos) in (argument)	var
参数替换	update (var1) at (pos) to (var2) in (argument)	var
二元操作符替换	update (op1) at (pos) to (op2)	op
变量类型替换	update (type1) at (pos) to (type2)	type
空值检查	insert if((var) ! =null) at (pos), move (statement) from (pos1) to (pos2)	var statement
返回语句添加	insert return (value) at (pos)	value
方法调用替换变量	delete (var) at (pos), insert (method) at (pos)	var, method

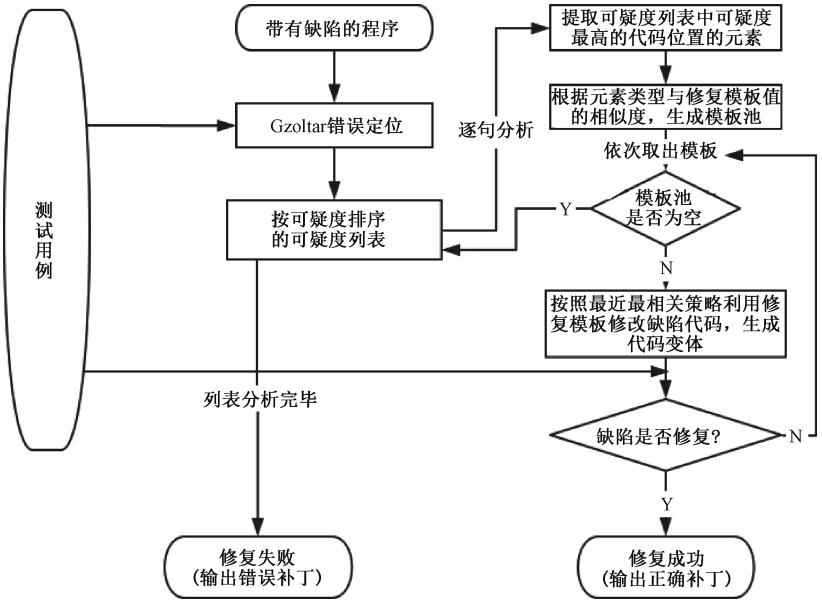

图 1　基于模板的程序自动修复框架

Figure 1　Template-based program automatic repair framework

了一种程序自动修复方法,其框架如图 1 所示。该方法利用常用的缺陷定位工具 Gzoltar[9] 结合测试用例给出代码语句的可疑度列表,针对可疑度列表中的各可疑缺陷代码语句,根据缺陷代码类型分析其可能适用的修复模板,得到模板池。再从模板池中依次取出模板,依据最近最相关策略选择模板操作元素,修改缺陷代码,生成代码变体。最后验证代码变体能否通过测试用例,以确定程序缺陷是否被成功修复。

下面详细讨论修复模板的选择、最近最相关策略以及程序自动修复。

2.1　修复模板的选择

利用缺陷定位工具 Gzoltar 可将缺陷定位到代码行,但无法定位此代码行产生缺陷的元素。APRMT 结合精确编译错误信息给出的元素信息,精准定位到缺陷代码位置。如表 1 所示,每个修复模板都具有其模板操作元素。APRMT 根据缺陷代码与缺陷模板元素的相似度生成多个修复模板,按其相似度的高低,生成模板池,并依次对缺陷程序进行修复,直到将程序缺陷成功修复,或者修复时间达到上限为止。

2.2　最近最相关策略

APRMT 在运用修复模板修改缺陷代码的抽象语法树并生成缺陷代码变体时,修复模板中目标模板操作元素的搜索空间是缺陷程序本身。本文提出了一种最近最相关策略,从缺陷程序源代码中搜索可用于该缺陷修复的代码元素。

最近最相关策略是指运用从模板池中取出的

修复模板对缺陷代码抽象语法树进行修改时,缺陷目标的修复代码元素的选择依据。修复模板在进行更新(update)、删除(delete)、移动(move)以及插入(insert)操作时,修复目标模板操作元素从原程序中,按照与缺陷代码距离最近、相似程度最高的原则进行挑选。这样既保证了无外来元素的添加,还能提高修复模板的修复效率。例如,APRMT利用Gzoltar与编译定位到的具体缺陷代码与缺陷模板操作元素的相似程度,匹配参数替换模板。但是参数在上下文中会有众多的选择,无法确保哪个参数是能够修复缺陷的,如果采用枚举策略,时间成本会非常高,修复缺陷的效率很低。因此,APRMT提出最近最相关策略,可减少时间成本,提高修复效率。

2.3 程序自动修复的实现

APRMT的实现借助了Astor[10]工具。Astor是一个基于验证的程序自动修复框架,其中包含了Gzoltar缺陷定位与Defects4J[11]的数据集。该框架提供的可扩展点包括:故障定位算法、修改点选取策略、操作符选择策略、修改点的粒度、成分池定义、操作符空间定义、检索策略。将APRMT扩展到Astor中,可采用:①故障定位算法,使用Gzoltar库的故障定位算法;②修改点选取策略,对于APRMT,缺陷代码行即修改点,优先选取可疑度最高的代码行作为修改点;③操作符选择策略,对于APRMT,一个修复模板就是一个操作符,根据所选的修改点选择相应的操作符。

将APRMT扩展到Astor中需要从以下4个方面进行改进。

(1)修改点的粒度。Astor涉及的修改点包括语句粒度级别、表达式级别和二元操作符级别。本文根据挖掘出的11个模板,对Astor从方法调用级别、变量级别和变量类型级别进行扩展。例如jGenProg[12]方法使用插入操作时,修改点为语句,作为插入的成分也是语句。但对于APRMT中的模板,修改点和修复成分的粒度可以不相同。例如,参数添加操作符的修改点粒度为方法调用级别,但插入的成分粒度为变量级别。因此在此处进行一个特殊的扩展,直接在Astor原项目代码中进行修改,使修改点的成分粒度与用于生成代码变体的成分池粒度区分开来。

(2)成分池定义。成分池与修改点粒度相对应,Astor原有的成分池包括语句池、表达式池和二元操作符池。根据扩展的修改点粒度,对成分池进行相应扩展,添加了调用方法池和变量池。

(3)操作符空间定义。对于APRMT,一个修复模板即一个操作符。本文将表1所示的11个修复模板作为操作符扩展到Astor中。

(4)检索策略。Astor仅针对给定粒度的可疑代码元素创建修改点,操作符从给定粒度的成分池中选择出现次数最多的成分,对修改点进行修改。在APRMT中采用最近最相关策略,利用可疑代码元素,在成分池中选择合适的成分,作为程序修复的代码。

3 实验

3.1 数据集

本文选取Defects4J上的数据集作为被测程序进行实验验证,如表2所示。由于程序自动修复需要利用测试用例进行错误定位,并验证修复是否成功,因此选取提供JUnit[13]测试用例的项目。表2中列出了各项目含有的缺陷数及其测试用例个数。

表2 Defects4J缺陷程序集合参数

Table 2 Defective program parameters of Defects4J

项目	缺陷数	测试用例个数
Chart	26	2 205
Lang	65	2 245
Math	105	3 602
Time	27	4 130

3.2 实验环境

APRMT借助Astor框架实现程序的自动修复。APRMT利用Gzoltar定位到缺陷语句所在的行,结合编译错误定位到缺陷代码所在的列,给出缺陷代码具体的位置信息。再按照可疑度排序依次取出缺陷代码,利用模板选择规则生成该缺陷对应的模板池。最后按照最近最相关策略从原缺陷程序中搜索目标模板操作元素进行缺陷代码的修复,利用测试用例验证是否成功修复。本文实验在Windows10操作系统上运行,每个程序缺陷的修复时间设为60 min,最大修复次数为100次。

3.3 实验结果与分析

APRMT使用的模板是从StackOverflow中挖掘的11个修复模板,见表1。

不同修复工具对Defects4J的修复结果如表3所示,其中数字代表该项目上修复成功的缺陷个数。修复成功是指测试用例在修复后的代码上执行通过,且人工验证了该修复结果与目标修复结果语义一致。

表 3　不同修复工具修复结果

Table 3　Repair results of different repair tools

项目	APRMT	SOFIX	Nopol	HistoricalFix
Chart	5	5	1	2
Lang	6	4	3	7
Math	12	13	1	6
Time	3	1	—	1

由表 3 可以看出,APRMT 成功修复了 26 个缺陷,共耗时约 80 min,平均每个缺陷修复时间为 3.1 min,由于缺陷定位不属于程序自动修复,所以将修复缺陷的开始时间设置为缺陷定位完成之后,未将错误定位消耗的时间计入修复缺陷时间内。

SOFIX 修复了 23 个缺陷,主要原因是其模板挖掘过程中抽象语法树的分析工具丢失了部分修改操作,造成挖掘的修复信息不准确,且 SOFIX 采用枚举方法进行修复,可能导致无法在一定时间内完成程序修复,甚至部分缺陷的修复需要生成 4 196 个代码变体并逐个验证,单纯修复这一个缺陷就需耗费大约 2 h,虽然最后将缺陷修复,但是却失去了程序自动修复的意义。Nopol[14] 的修复仅针对 if-then-else 语句中存在的缺陷,因此其只修复了 5 个缺陷,平均每个耗时 9 min,但是其正确修复的缺陷数量及类型限制了程序自动修复的效率。HistoricalFix[15] 是在假定缺陷位置已知的情况下进行的程序修复,这将降低程序修复难度,但其平均每个缺陷的修复时间为20 min,修复效率不高,而且 HistoricalFix 收集到的人工修复信息具有一定的局限性[16]。

此外,为了证明 APRMT 的有效性,与基于遗传规划的程序自动修复工具 jGenProg 做了对比实验。如表 4 所示,APRMT 和 jGenProg 对 8 种缺陷都修复成功,但在时间成本方面,jGenProg 大约是 APRMT 的 4 倍;在修复效果方面,APRMT 由于结合了修复模板中的修复信息,只需生成 135 个补丁便可完成程序自动修复,而 jGenProg 则需 762 个补丁,是 APRMT 的 5.6 倍。由此可见,APRMT 无论是在时间效率上,还是修复效果上都远超 jGenProg。

由上述分析可知,APRMT 的修复效率是比较稳定的,主要原因有 2 个方面:一是在修复模板挖掘的过程中,APRMT 关注的修复信息不局限于 if-then-else 条件语句,并且对修复信息进行预处理,尽可能挖掘更多的修复模板,保证修复模板的有效性与完整性,增加了修复缺陷个数;二是在程序自动修复过程中,APRMT 结合最近最相关策略,搜索目标修复模板操作元素,避免了枚举策略的耗时,提高了程序修复效果和效率。

表 4　APRMT 与 jGenProg 的对比实验结果

Table 4　Results of the APRMT vs. jGenProg.

缺陷所在类	耗时/s		生成补丁数	
	APRMT	jGenProg	APRMT	jGenProg
NEXT_PALINDROME	135	1 320	17	188
GCD	57	732	8	85
WRAP	289	1 042	42	206
SUBSEQUENCES	24	206	2	31
BUCKETSORT	139	621	24	105
SUBSEQUENCES	431	876	27	73
BITCOUNT	43	82	2	18
FIND_REPEAT_NUM	94	143	13	56

4　结论

本文提出了一种改进的基于模板的程序自动修复方法 APRMT。在修复模板挖掘中,APRMT 采用正则匹配消除修复信息中的无用信息,分析抽象语法树时,对差异节点进行跟踪,避免了修改动作的缺失,提高了挖掘修复信息的准确性与完整性。在程序自动修复中,APRMT 利用最近最相关策略,搜索需要修复的目标模板操作元素,实现程序自动修复。实验表明,APRMT 能在较短的时间内修复 Defects4J 上的 26 个缺陷,程序自动修复的效果与效率有了较大提升。

参考文献:

[1] WEIMER W, NGUYEN T, LE GOUES C, et al. Automatically finding patches using genetic programming[C]// 2009 IEEE 31st International Conference on Software Engineering. New York: IEEE, 2009: 364-374.

[2] MARTINEZ M, MONPERRUS M. Astor: exploring the design space of generate-and-validate program repair beyond GenProg[J]. Journal of systems and software, 2019, 151: 65-80.

[3] KIM D S, NAM J C, SONG J, et al. Automatic patch generation learned from human-written patches[C] // Proceedings of the 2013 International Conference on Software Engineering. New York: ACM, 2013: 802-811.

[4] LIU X L, ZHONG H. Mining stackoverflow for program repair[C]//2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). New York: IEEE, 2018: 118-129.

[5] LE X B D, LO D, LE GOUES C. History driven program

repair[C]//2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER).New York:IEEE,2016:213-224.
[6] XUAN J F,MARTINEZ M,DEMARCO F,et al.Nopol: automatic repair of conditional statement bugs in Java programs [J]. IEEE transactions on software engineering,2017,43(1):34-55.
[7] MARTINEZ M, MONPERRUS M. Mining software repair models for reasoning on the search space of automated program fixing[J].Empirical software engineering,2015,20(1):176-205.
[8] FALLERI J R,MORANDAT F,BLANC X,et al.Fine-grained and accurate source code differencing[C]// Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering.New York: ACM,2014:313-324.
[9] CAMPOS J,RIBOIRA A,PEREZ A,et al.GZoltar: an eclipse plug-in for testing and debugging[C]//2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering.New York: IEEE,2012:378-381.
[10] SOBREIRA V,DURIEUX T,MADEIRAL F,et al.Dissection of a bug dataset: anatomy of 395 patches from Defects4J [C]//2018 IEEE 25th International Conference on Software Analysis,Evolution and Reengineering (SANER).New York:IEEE,2018:130-140.
[11] PAWLAK R,MONPERRUS M,PETITPREZ N,et al. SPOON:a library for implementing analyses and transformations of Java source code[J].Software: practice and experience,2016,46(9):1155-1179.
[12] MARTINEZ M, MONPERRUS M. ASTOR: a program repair library for Java(demo)[C]//Proceedings of the 25th International Symposium on Software Testing and Analysis. New York: ACM, 2016:441-444.
[13] HASSAN F, WANG X Y. HireBuild: an automatic approach to history-driven repair of build scripts[C]// Proceedings of the 40th International Conference on Software Engineering.New York:ACM,2018:1078-1089.
[14] JUST R,JALALI D,ERNST M D.Defects4J:a database of existing faults to enable controlled testing studies for Java programs[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. New York:ACM,2014:437-440.
[15] TANG J W,LI R X,WANG K P,et al.A novel hybrid method to analyze security vulnerabilities in Android applications[J]. Tsinghua science and technology, 2020,25(5):589-603.
[16] 张茂清,汪镭,崔志华,等.基于混合策略的快速非支配排序算法Ⅱ[J].郑州大学学报(工学版),2020,41(4):23-27.

Automatic Program Repair Method Based on Template Mining

HAN Junxuan, SUN Weifeng, ZHAO Ruilian, WANG Weiwei

(School of Information Science and Engineering, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract: In the process of program repair template mining, due to the noise interference in the historical repair information, the availability of the mined repair template was not strong, and the program repair effect is not good. In order to improve the efficiency of automatic repair. It was an important method use repair templates to repair the program. Therefore, from the perspective of repair template mining and automatic program repair, an improved program automatic repair method is proposed, called APRMT (automatic program repair method based on template mining). APRMT can eliminate the noise of repair information through regular matching, and the accuracy of repair information. According to the location and type of the program defect code to search and repair the information of the defect, the most recent strategy was proposed, which could efficiently use the repair template to repair the program defect. In order to verify the effectiveness of this method, APRMT experimented on 4 open source projects, and the experimental results showed that APRMT successfully fixed 26 defects, with an average time of 3.1 minutes per defect. It can be seen that APRMT can fix more program defects in a shorter time, and therefore improved the effect and efficiency of automatic program repaired.

Key words: automatic repair; template; program; defect; mining