### 软件自动修复文献综述

摘要：尽管现代软件应用程序越来越复杂，规模也越来越大，但它们必须满足严格的发布要求，给修复错误和维护的周期很短，这对开发者开发高质量软件带来了巨大的压力。为了减小开发者的工作量和节省软件开发成本，近年来软件缺陷自动修复引起了广泛地关注并且大量的修复技术被研究。一系列算法、技术被研究出来并应用于软件缺陷自动修复领域。本文通过调查一系列软件自动修复领域的文献，对本领域的知识进行整理，说明了该领域基本的概况。

关键字：软件缺陷、oracle、生成与验证、补丁、测试集

# 研究背景和意义

软件缺陷（software defects）在软件开发过程中不可避免。随着当代信息技术的迅速发展，软件的规模和数量也在增长，但是软件缺陷的数量也随之增长。软件缺陷会破坏程序的正确执行，使得程序无法按照预期的行为输出正确的结果。严重的软件缺陷会造成社会巨大财富损失。如1996年6月4日，Ariane5火箭由于浮点数溢出导致解体并爆炸，造成至少五亿美元损失；加拿大原子能公司的therac-25放射性治疗仪，由于系统软件缺陷导致病人遭受过量辐射，最终造成4人死亡两人重伤的特大医疗事故。根据美国国窖标准与技术研究院估算，软件缺陷每年给美国带来的经济损失高达595亿美元。因此，及时修复软件的故障成为一件非常重要的任务。研究表明，修复软件缺陷竟花费软件开发过程中一半的时间和资源[1]。北美工业界统计调查发现，北美公司平均每年花费2200万美元来修复软件缺陷。而在2001年到2010年，Eclipse共接受了333371个提交的bug，平均每天新增76个bug。所以修复软件缺陷不仅花费大量时间，而且往往会引入新的错误，这使得软件缺陷修复变得更加困难。

软件缺陷自动修复（automatic software repair）技术有助于开发人员从繁重的缺陷定位、缺陷修复的任务中解放出来。该技术根据给定的软件错误和测试集，自动生成补丁，并且生成的补丁可以自动添加到程序也可以指导开发人员继续改进代码。软件自动修复技术将有效地降低程序修复的成本。自从2009年的GenProg提出以来，大量研究人员投入到这个研究领域中，提出了许多技术和算法：启发式搜索、深度学习、软件分析等。虽然有大量的技术被应用于软件自动修复，但是自动修复技术对软件缺陷信息挖掘不深，只能修复简单的错误，难以修复复杂的错误，并且生成的补丁还存在过拟合的问题，即使通过了测试集，却仍然是错误的。提高修复能力是该领域的重要内容。

# 国内外研究现状

软件自动修复一般分为三个部分：缺陷定位(fault localization)、补丁生成(patch generation)、补丁过滤和排序。

根据修复方法的不同，一般可以分为以下三大类：（1）基于模板的修复方法；（2）基于搜索的修复方法；（3）基于语义的修复方法。第一类主要是根据人工的补丁集作为数据源，提取出的修复模板。然后将模板应用与对应的软件缺陷。第二种主要是通过启发式搜索，从程序本身出发寻找正确的补丁。第三种通过将生成补丁转化为约束求解问题，通过运行测试集得到约束应用求解器求解。这三种方法是当前领域主流的修复方法。

## 2.1 基于人工修复模板

（1）利用模板生成补丁

基于人工模板生成补丁指的是开发人员根据经验预先定义一些补丁模板用于修复，当程序的环境满足某一个模板时，利用模板直接生成补丁。2013年提出的PAR是代表性研究，Kim等人预先定义了10个修复模板，其中包含了6个大类的代码修复，比如替换函数参数、修改变量初始化等。2019年，Koyuncu等人提出将PAR用来修复缺陷报告中的缺陷，并且提出了修复工具iFixR。但是人工定义的模板难以覆盖所有缺陷，所以还有人提出了针对特定软件缺陷的修复工具。比如2019年Marginean提出的修复工具SapFix主要针对面向对象语言中的空指针缺陷等。

除了人工定义的模板外，利用代码自动提取模板也是一个重要的方法。该方法利用原先已有的补丁库，提取出模板用于生成补丁。2020年，Kim等人提出CCA工具。该工具通过从代码库中收集抽象语法树的结构，不断地提取不同精度的语法树，直到获取到最抽象的模板，然后按相同的方法应用到bug程序中生成补丁。2019年，facebook提出的Getafix被应用与实际开发。Getafix通过对抽象语法书识别修改，保存具体的修复，而且采用新的分层聚类技术，生成的模板不仅仅是代码的更改，而且还保存了上下文以便更好地匹配模板。通过一次次泛化，从具体到抽象，能够为软件缺陷生成最接近人工的补丁，从而大大提高开发人员的接受度。

（2）利用补丁模板优化候选补丁

与PAR相反,2016年,Tan等人通过分析修复工具GenProg和SPR所产生的修复补丁发现,自动修复工具产生的大部分不正确补丁具有公共的特点:删除了程序中的部分代码语句.基于上述发现,作者提出了反模式 (anti-pattern)方法.该方法在已有的自动修复工具基础之上定义了一系列规则对工具产生的补丁进行过滤,比如 不能删除返回语句(return)、不能删除条件语句(if)等.作者基于GenProg和SPR自动修复工具实验,对其增加了上述补丁过滤条件.实验结果表明,上述过滤规则可以有效避免不正确的修复补丁。此外,该方法由于删除了部分候选补丁,补丁验证时间减少,修复的效率平均提升了1.4倍。但相比原始方法,过滤规则并没有使修复工具产 生新的修复补丁,修复工具的召回率依然由其自身所决定,反模式只是在一定程度上提升了补丁的准确率。

## 基于语义约束

基于语义约束的自动修复方法通过某种手段推断程序的正确规约,作为约束指导补丁的生成过程或对补丁的正确性进行验证.比较有代表性的研究工作是新加坡国立大学 Roychoudhury教授团队在2013年和2015年先后提出来的自动修复工具SemFix和 DirectFix,这两种方法均通过符号执行技术在缺陷代码位置搜集 使测试通过的程序语义约束信息,然后使用基于组件(component-based)的程序合成方法生成修复补丁.该过程 需要依赖约束求解器(SMT solver)求解.DirectFix相比于前者对补丁生成过程进行了优化:DirectFix不严格按照 定位的结果,而是优先选择比较简单的候选语句进行修复;其次,DirectFix将补丁生成问题转换为部分最大可满足问题(partial MaxSAT),通过将补丁的复杂度(即对原始代码的修改大小)作为软约束(soft condition)来控制pMaxSMT求解器,生成更简单的修复补丁.

上述介绍的补丁生成方法通过重新组合项目中的已有代码片段生成修复补丁.近期的一些研究通过编码程序的语义信息,搜索可以直接复用的代码片段用于生成补丁代码.与MemFix类似,2018年,van Tonder和Le Goues提出了 FootPatch,针对Java语言的资源泄漏、内存溢出和空指针引用等缺陷.在该方法中,FootPatch 将程序的语义约束信息用分离逻辑(saparation logic)描述作为程序的规约.在最后的补丁生成阶段,FootPatch 在 项目中搜索满足上述规约的代码段,直接复用其作为正确修复补丁.

## 基于统计分析

（1）补丁排序模型

基于统计分析的缺陷修复技术指利用统计的方法或学习的技术指导补丁生成的过程或对候选补丁进行优化.比较早的相关工作是2016年由Long和 Rinard提出来的修复方法Prophet,该方法是在SPR基础之上的进一步优化.SPR 也是Long等人在2015年提出来的基于人工定义模板的C语言条件语句修复技术.Prophet根据修复前后的代码特征训练了一个排序模型对 SPR 生成的修复补丁进行排序.Prophet搜集程序的两类特征信息:首先是操作,即修复补丁所使用的代码修改有哪些,例如插入语句或替换语句等;另一类特征是程序的静态 特征,例如包含循环语句或检查某个变量的取值等.通过实验表明,该方法可以有效过滤不正确的修复补丁.虽然 Prophet使用SPR同样的补丁生成技术,却可以多修复一个缺陷.但该结果也表明,单纯通过补丁排序技术难以实现较大的效果提升.

（2）提取补丁模板用于修复

2017年,Xiong 等人针对Java语言中的条件语句修复,提出了高精度的条件语句综合技术ACS.该技术应用了3种技术手段优化生成补丁的质量:首先,ACS利用代码的局部性原理,根据变量之间的依赖拓扑关系对修复条件中的变量选取进行排序,在拓扑图中,距离代码修改位置比较近的变量优先使用;此外,ACS通过分析代 码中的自然语言注释内容,挖掘程序特征来补充测试所定义的不完整程序规约;最后,ACS通过统计分析相似项目中常用的条件谓词,用来对修复补丁所使用的表达式类型进行筛选和排序.通过分析其他项目中的频繁谓词 可以获得领域特定知识,例如变量“hour”通常与常量12进行比较等,从而有效提升补丁的质量.实验结果表明: 该方法相比之前方法,实现了大概两倍的修复准确率.

## 基于启发式搜索

（1）利用变异算子

基于启发式搜索的自动修复技术通过人工定义启发式规则,指导修复补丁的生成过程.2009年被提出来的缺陷自动修复技术GenProg是典型的基于启发式搜索的技术,其基本思想是代码具有重复性,因此通过复 用项目中的代码,有希望产生正确的修复补丁.在搜索过程中,GenProg采用遗传算法,通过定义代码片段的交叉 和变异操作实现已有代码片段的重新组合,增大补丁的搜索空间.该方法将生成补丁所通过的测试数量作为优化目标,对候选补丁迭代应用交叉和变异操作,直至产生可以通过所有测试的补丁.通过实验证明,该方法可以修复程序中的缺陷.该工作使得自动修复技术进入了一个新的时期,吸引了大量的科研人员开始对自动修复技术进行探索.

2018年,Yuan等人提出了基于多目标遗传算法的自动修复方法 ARJA,该方法同样是对遗传算法中的表示方法进行了优化.ARJA 将代码补丁表示为三元组〈b,u,v〉,分别用来编码代码上的修改位置、修改操作类型和复用的代码元素.该方法由于记录了代码的修改过程,因此具有更强的表达能力,通过变异操作可以获得更大的补丁空间.除此之外,ARJA将补丁生成过程转化为多目标优化问题,其两个优化目标分别是补丁对代码改动的大小以及应用补丁后程序通过测试的数量.因此,该方法倾向于修改较小的修复补丁.最后,ARJA在补丁验证阶段通过移除与当前补丁无关的测试以及通过编译器分析提前排除不合法的补丁等方式提升补丁验证的效率.最终实验结果表明,ARJA正确修复缺陷的数量接近GenProg的4倍.

（2）利用历史修复补丁

与上述基于遗传算法的缺陷修复不同,Le等人认为:在不同的应用软件中,软件缺陷会重复出现,之前的缺陷修复历史可以为修复补丁提供有效的指导.因此,引入了第三方数据源(即历史修复)对GenProg进行优化.基于该思想,作者在 2016 年提出一个新的缺陷修复技术HistoricalFix.该方法同样采用了遗传算法来生成修复补丁,不同的是,HistoricalFix仅仅采用变异(mutation)而不使用交叉(crossover)操作.在每次变异之后,计算候选补丁所涉及的修改操作,根据其他项目历史修复中常用的代码修改对补丁进行筛选.该方法相比之前的遗传算 法引入了第三方的缺陷修复历史数据,对补丁的生成起到了更好的指导作用.最终的实验证明,该方法相比原始的GenProg方法有了非常大的效果提升.在 Defects4J数据集中的90个缺陷上进行验证,GenProg只能正确修复1个缺陷,而HistoricalFix可以正确修复23个.该实验表明,历史修复数据可以为补丁生成提供有效指导.

（3）利用相似代码

上述技术主要针对补丁的搜索策略和代码的修改进行优化,还有一部分最新研究关注于优化补丁生成过程中所使用的代码元素.上述介绍的方法在复用项目中已有的代码时,仅考虑代码出现的频繁程度(例如CapGen),甚至完全不考虑代码的特征,并且均没有考虑缺陷代码与复用代码之间的联系.针对于此,2016 年,Ji等人提出通过复用项目中与缺陷位置相似的代码片段生成补丁,并实现了缺陷修复工具SCRepair.作者认为,与缺陷代码过于相似的代码可能存在同样的错误.因此,SCRepair在复用相似代码作为修复参考时考虑了代码之间需要具有差异性.SCRepair通过对比缺陷代码与参考代码之间的语法树结构的一致性衡量代码相似性,使用 ChangeDistiller提取缺陷代码到参考代码的修改操作衡量代码之前的差异性.根据预设的相似性和差异性阈值对复用的参考代码进行过滤.SCRepair与RSRepair相似,差别在于SCRepair在复用代码时考虑上述过滤条件,而RSRepair采用无过滤的随机搜索.

## 过拟合补丁的问题

自动程序修复(APR)在过去的十多年里引发了人们的广泛研究，大量的修复技术被提出。特别是GenProg提出以来，基于测试套件的修复技术占据了其中的大部分。基于测试套件的修复工具以给定的测试套件为oracle，生成的补丁如果能够通过测试套件，那么补丁将被视为正确的。但是在实际过程中，测试套件是弱的，并不能完全表达出程序的oracle，能够通过所有测试用例的补丁并不是完全正确的，存在过拟合的问题。这就导致了APR技术产生了大量的无效的补丁。目前的修复技术远远不够成熟，绝大部分修复技术只会简单接受通过测试套件的补丁。根据，Genprog，AE，RSRepair等技术大部分生成的补丁都是错误的，无法被人接受。即使是采用了其他的技术(e.g., 使用修复模板和条件合成，bug-fixing instances 和forbidden modifications)，仍具有较低的修复性能。

为了过滤出这些补丁，开发人员往往需要手动验证补丁，这耗费了太多资源。由于修复技术的低性能，开发人员必须验证大量的错误补丁。解决补丁的过拟合问题成为一个需要研究的领域。如果一个补丁通过测试套件就被认为是正确的补丁，那么只要增强测试套件就能够减少过拟合补丁的数量。但是自动测试生成工具只能生成测试输入，如何确定合适的测试输出仍需要人类手动确定。这样的做法仍然无法表达出完整的oracle。想要一个程序有完整的oracle是十分困难的，尤其是大型项目，Nilizadeh使用formal methods (specification and verification) 作为正确性标准，用JML来描述自己的小型程序集十分成功，但是在面对大型项目defects4j时，JML就失效了，无法用JML来描述如此大的项目。

过拟合补丁不仅仅耗费开发人员的时间，在此基础上，过拟合补丁还存在好几种类型。Yu通过检测测试输入的范围将过拟合补丁分为三类：

1. A-Overfitting Patch：补丁没有完全修复不正确的行为以及没有破坏原来正确的行为。
2. B-Overfitting Patch：补丁修复了原来不正确的行为但是破坏了原来正确的行为。
3. AB-Overfitting Patch：补丁既没有修复不正确的行为也破坏了原来正确的行为。

在目前，能够识别出一个补丁是否过拟合已经是很成功的事了。因为快速识别过拟合补丁能够提高APR技术和开发人员修复错误的成功率，但是如果有技术能够对过拟合补丁再细分，那么能够大大提高开发人员修复程序错误的速度。

由于存在过拟合补丁，人类也开始逐步改变修复策略，仅仅程序的输出符合测试输出是远远不够的，也就是说当前APR技术对测试套件的利用率太低了，只利用了测试套件的测试输入和测试输出，并没有深层次地挖掘隐藏在测试套件中的信息。除此之外也有好几种过拟合检测技术被提出来，其中Xiong提出的策略是第一种挖掘测试套件和程序深层行为的技术，通过TEST-SIM和PATCH-SIM的思想，能够使过拟合补丁排除的成功率达到56%。而Yang的实验说明了补丁会修改程序的行为，他们试图利用程序不变量来描述程序的行为，从另一个角度来观察程序的Oracle。