回归测试中测试用例集扩充技术研究进展

陈 翔[1,2] 顾 庆[2] 陈道蓄[2]

(南通大学计算机科学与技术学院 南通 226019)[1] (南京大学软件新技术国家重点实验室 南京 210093)[2]

摘 要 测试用例集扩充问题(Test Suite Augmentation)是回归测试研究的一个最新研究热点。在完成代码修改影响分析后,对已有测试用例集的充分性进行评估,若不充分则设计新的测试用例,以确保对代码修改的充分测试。但到目前为止,国内外学者并未对该研究问题的已有研究成果进行系统总结和展望。首先介绍了测试用例集扩充问题的研究背景和问题描述,然后总结出研究框架并对已有研究工作进行分类和系统比较,接着对常用评测数据集和评测指标进行了分析,最后对该问题值得关注的未来研究方向进行展望。

关键词 回归测试,测试用例集扩充,测试覆盖准则,动态符号执行,演化测试

中图法分类号 TP311.5 **文献标识码** A

Research Advances in Test Suite Augmentation for Regression Testing

CHEN Xiang[1,2] GU Qing[2] CHEN Dao-xu[2]

(School of Computer Science and Technology, Nantong University, Nantong 226019, China)[1]

(State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210093, China)[2]

Abstract Test suite augmentation is a hot research issue in regression testing. After code change impact analysis, we evaluated the adequacy of existing test suite. If not adequate, we will design new test cases to adequately test the code change. Until now, no researchers have given a survey for this research topic. We firstly introduced the research background and problem description of this topic. Secondly we summarized a framework of this topic and made a systematic classification and comparison for existing research work. Thirdly we introduced the common used experiment subjects and evaluation metrics. Lastly we gave some suggestion on potential future work for this research area.

Keywords Regression testing, Test suite augmentation, Test coverage criterion, Dynamic symbolic execution, Evolutionary testing

1 引言

软件产品在开发和维护过程中因代码修改会触发软件演化行为,引发代码修改的主要原因包括:移除软件内部已有缺陷、完善软件提供的功能、重构软件已有代码或提高软件产品的运行性能等。随着一些新的软件开发方法和模型(如敏捷软件开发、极限编程和测试驱动开发等)的日益成熟和不断推广,软件演化频率也迅速提高并亟需经济有效的方法来确保演化后软件产品的质量。为了有效保障修改后的软件产品的质量,测试人员一般通过执行回归测试来保证代码修改模块的正确性并避免代码修改模块对被测程序其他模块产生的副作用,从而在软件交付阶段提高对开发出的软件产品的信心。但目前有统计数据表明回归测试所需开销一般占整个软件测试预算的80%以上,并且占整个软件维护预算的50%以上[1,2]。为了降低回归测试的开销,国内外研究人员对自动高效的回归测试技术展开了深入研究并取得了丰硕的研究成果。其中对测试用例的有效维护是回归测试中的一个重要问题。一种简单维护策略是重新执行已有的所有测试用例,但该策略存在如下问题:(1)在一些测试场景中,若测试用例的数量较多或单个测试用例的执行开销较大时,则执行完所有测试用例后所需的开销将超出项目的实际预算。(2)部分代码修改会影响到被测程序的原有外部接口和内在语义,从而导致部分测试用例失效。(3)若代码修改产生新的目标测试需求,则需要额外设计新的测试用例来确保对代码修改的充分测试。针对上述问题,研究人员提出了一系列行之有效的测试用例维护技术,关键技术包括:失效测试用例的识别和修复技术、测试用例选择技术、测试用例优先级排序技术、测试用例集约简技术和测试用例集扩充技术等[3]。

与测试用例选择、测试用例优先级排序和测试用例集约简等问题相比,测试用例集扩充问题在近几年日益得到研究人员的关注,他们提出了多种有效解决方法。具体来说,首先通过对修改前后程序的对比获得代码修改信息。其次设计测试覆盖准则来指导对代码修改的充分测试。最后判断已有测试用例集是否满足该测试覆盖准则,若不满足则考虑设计新

到稿日期:2012-08-08 返修日期:2012-11-21 本文受国家自然科学基金(60873027,61202006),江苏省高校自然科学研究项目(12KJB520014),南通市应用研究计划项目(BK2012023),南京大学计算机软件新技术国家重点实验室开放课题(KFKT2012B29)资助。

陈 翔(1980—),男,博士,讲师,主要研究方向为软件测试和程序分析,E-mail:xchencs@ntu.edu.cn;顾 庆(1972—),男,博士,教授,博士生导师,主要研究方向为软件质量保障;陈道蓄(1947—),男,教授,博士生导师,主要研究方向为分布式计算与并行处理。

的测试用例。虽然国内外研究人员对回归测试中的一些重要技术均发表过综述论文，例如：在国外，Yoo 和 Harman 对回归测试中的测试用例约简、选择和优先级问题进行了系统总结[4]；在国内，章晓芳等人对测试用例集约简问题进行了总结[5]，而屈波等人则对测试用例优先级问题进行了总结[6]。但是到目前为止，国内外研究人员均未对测试用例集扩充问题进行系统分析、总结和展望。

我们通过对 IEEE、ACM 和 Spring 等数据库的检索，并对近些年与软件工程研究领域相关的会议和期刊发表的论文进行筛选，从候选论文中最终确定了与综述主题直接相关的 12 篇论文(截止到 2012 年 8 月份)。图 1 按照论文发表的时间进行汇总，其中横坐标表示发表年份，纵坐标表示该年份发表的论文总数。从图 1 可以看出，近 5 年发表的论文数占总数量的比例高达 75%。表 1 则对 12 篇论文出版的会议或期刊进行汇总。从表 1 不难看出，绝大部分论文发表在软件工程的顶级会议和顶级期刊上，例如在 FSE(International Symposium on Foundations of Software Engineering)会议上发表 2 篇，在 ISSTA(International Symposium on Software Testing and Analysis)会议上发表 2 篇，在 ASE(International Conference on Automated Software Engineering)会议上发表 2 篇，在 TSE(IEEE Transactions on Software Engineering)期刊上发表 1 篇。其中 Santelices 等人的研究工作[7]甚至获得 ASE 2008 的最佳论文奖。从图 1 和表 1 中可以看出，该研究问题在近几年内逐渐得到国内外研究人员的关注，并通过在顶级会议或期刊上的发表将会吸引更多的国内外研究人员对该问题展开更为深入的研究。

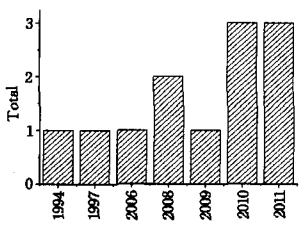

图 1　论文在不同发表时间的数量分布

表 1　对已有论文按发表会议和期刊进行汇总

会议或期刊简称	类型	总数	相关论文
FSE	会议	2	徐志弘等[8]、Person 等[9]
ISSTA	会议	2	Rothermel 等[10]、Taneja 等[11]
ASE	会议	2	Santelices 等[7]、齐大伟等[12]
ICST	会议	1	Santelices 等[13]
ISSRE	会议	1	徐志弘等[14]
GECCO	会议	1	徐志弘等[15]
APSEC	会议	1	徐志弘等[16]
TAIC PART	会议	1	Apiwattanapong 等[17]
TSE	期刊	1	Binkley[18]

通过上述分析，不难看出测试用例集扩充问题是目前回归测试的一个研究热点。本文首次对该问题的已有研究工作进行系统总结和比较；第 2 节对测试用例集扩充问题的研究背景和问题描述进行简述；第 3 节给出一个研究框架，并在框架内对目前已有的研究成果进行总结和系统比较；第 4 节和第 5 节分别分析了标准评测数据集和评测指标；第 6 节对该研究点的下一步可能研究方向进行展望；最后对全文进行总结。

2　研究背景

2.1　回归测试

回归测试是软件开发和维护阶段保障软件产品质量的重要手段。通过对已有研究工作的分析和总结，我们提出了图 2 所示的研究框架。首先对修改前后程序进行建模和比对以识别出代码修改模块，通过代码修改影响分析可以识别出与代码修改相关的测试需求。随后在已有的测试用例集基础上通过依次执行如下的测试用例维护活动可以有效地保障修改后的软件产品质量。(1)无效测试用例的识别和修复技术：软件修改一般会导致相关模块的外部接口或内在语义发生变更，从而造成部分测试用例无效。对于识别出的无效测试用例，若直接进行移除会显著降低原有测试用例集的质量，目前一种可行的方法是对这些测试用例进行修复[19,20]。(2)测试用例选择技术：给定修改前后程序和代码修改信息，测试用例选择技术通过从已有测试用例中选择出与代码修改相关的测试用例来完成修改后程序的测试，并确保未被选择的测试用例在修改前后程序上的执行行为保持一致，一种常见的方法是图遍历法[4,21-23]。(3)测试用例集扩充技术：该问题将在 2.2节给予详细阐述。(4)测试用例集约简技术：当完成上述测试用例维护行为后，根据指定的测试覆盖准则(例如判定覆盖准则)，可以进一步识别并移除冗余测试用例[4,5,24-30]。但测试用例集约简技术存在的主要问题是可能会移除一些具有缺陷检测能力的测试用例并降低原有测试用例集质量。(5)测试用例优先级排序技术：当测试预算不足以致不能执行完所有测试用例时，可以基于特定排序准则对测试用例的执行进行排序，期望通过排序可以最大化测试用例集的早期缺陷检测能力[4,6,31-35]。

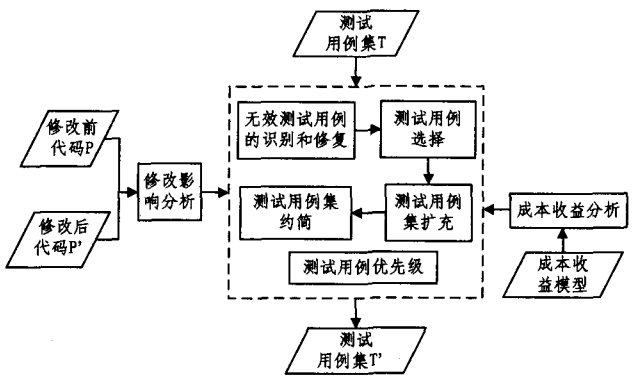

图 2　回归测试研究框架

将上述学术界的研究成果成功应用到软件企业的关键是进行成本收益分析，精确的成本收益分析离不开合理的成本收益模型。目前对模型内的影响因素主要从成本角度和收益角度进行分析，考虑的因素包括源代码修改分析、代码修改影响分析、测试用例运行环境准备、测试用例执行、测试用例结果检查、测试用例选择、约简、优先级等方法，以及因遗漏具有缺陷检测能力的测试用例造成的损失等[36-38]。

2.2　测试用例集扩充问题描述

虽然测试用例选择、约简和优先级排序技术已经得到了研究人员的广泛关注，但测试用例集扩充问题作为测试修改后程序的一个基本问题一直未得到研究人员应有的关注。该问题可以简要描述如下：

给定：修改前程序 P 和相应测试用例集 T，修改后程序 P'。

问题:判断已有测试用例集 T 对代码修改是否进行了充分测试,若未充分测试,则额外设计新的测试用例,以确保可以检测出修改前后程序上的执行行为差异,从而辅助测试人员发现与代码修改相关或与代码修改产生的副作用相关的程序缺陷。

目前已有的研究工作一般以 Voas 提出的 PIE 模型[39]为基础。在该模型中,若要测试用例 t 的执行使得包含缺陷的程序 p 产生失效,需要满足如下 3 个条件:(1)E 条件:测试用例 t 在执行时覆盖到了缺陷语句。(2)I 条件:测试用例 t 在执行完缺陷语句后造成了错误的程序内部状态。(3)P 条件:错误的程序内部状态通过传播对程序 p 的实际输出结果产生影响,即与测试用例的预期输出不一致。

研究人员一般借助 PIE 模型来指导与代码修改相关的测试覆盖准则的设定,因为测试用例集扩充技术主要关注可以覆盖到与代码修改相关的语句并可能对程序输出产生影响的测试用例。

测试用例集扩充问题作为一个研究热点,与其他回归测试问题存在紧密依赖关系,这里主要分析测试用例集扩充问题和测试用例选择问题及测试用例生成问题的关系。(1)首先分析测试用例集扩充问题与测试用例选择问题之间的关系。测试用例选择问题主要考虑从已有测试用例集 T 中选择出部分测试用例用于修改后程序的测试,并确保未被选择的测试用例在修改前后程序上的执行行为不会存在差异[4,21-23]。而测试用例集扩充问题不仅需要考虑对已有测试用例的复用,而且在大部分情况下还需要额外生成与代码修改相关的测试需求,即同时选择和生成测试用例来完成对这些测试需求的充分覆盖。(2)其次分析测试用例集扩充问题与传统测试用例生成问题间的关系。传统测试用例生成问题是软件测试研究中的一个经典问题,测试用例集的充分性一般建立在测试覆盖准则基础之上,所以该问题首先根据被测软件特征设定新颖的测试覆盖准则并生成目标测试需求,这些覆盖准则的设定一般基于控制流分析或数据流分析[40],例如控制流分析关注的程序实体可能是语句(块)、分支等。数据流分析关注的程序实体可能是定义使用对等。其次设计测试用例生成技术完成对上述测试需求的充分覆盖,典型的测试用例生成技术包括随机测试法、动态符号执行法和演化测试法等。而测试用例集扩充问题则仅仅关注与代码修改相关的测试覆盖准则设定,这些覆盖准则的设定一般同时考虑传统的覆盖准则和 PIE 模型,并希望能够选择或构造出可以检测出在修改前后程序上执行行为存在差异的测试用例,从而判断代码修改的结果是否符合修改预期。

3 已有研究工作

在对已有文献的深入总结和分析基础之上,本文提出图 3 所示的测试用例集扩充技术框架。具体来说:首先对修改前后程序进行建模,其次识别出代码修改模块并进行修改影响分析,在分析基础上设计基于代码修改的测试覆盖准则,然后判断已有测试用例集 T 是否满足该测试覆盖准则,若不满足则设计新的测试用例并添加到 T 中。从该框架中可以看出测试用例集扩充技术主要包括两个部分:代码修改相关的测试覆盖准则的设定和新的测试用例生成算法。在这一节,将按照这两个部分对已有研究工作进行分类、比较和总结。

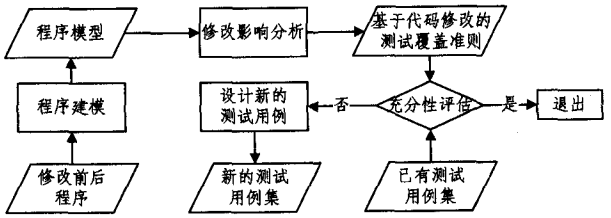

图 3 测试用例集扩充技术框架

3.1 代码修改相关的测试覆盖准则的设定

软件测试作为一种软件质量保障手段,仅能检测出被测软件内部存在的缺陷,而不能证明软件内部不存在缺陷。在实际的软件测试工作中,设计出覆盖被测软件所有可能输入的测试用例并不可行,一般通过设计测试覆盖准则来对测试用例集的充分性进行评估。针对测试用例集扩充问题,通过设定代码修改相关的测试覆盖准则,可以指导生成一系列测试需求,从而辅助测试人员对已有测试用例集的充分性进行有效评估。

Binkley[18]、Rothermel 和 Harrold[10]首先采用系统依赖图(System Dependence Graph)对被测程序进行建模,然后通过程序切片(Slice)识别出与代码修改相关的控制或数据依赖关系来设计相关测试覆盖准则。控制依赖关系、数据依赖关系和依赖链分别定义如下。

定义 1(控制依赖关系) 语句 $S1$ 控制依赖于语句 $S2$,当语句 $S2$ 至少有两条输出控制边并至少存在一条路径,该路径同时覆盖了语句 $S1$ 和 $S2$。

定义 2(数据依赖关系) 语句 $S1$ 数据依赖于语句 $S2$,当(1)变量 v 在语句 $S2$ 上定义出现,(2)存在从语句 $S2$ 到语句 $S1$ 的 definition-clear 路径(即在该路径上不存在变量 v 的定义出现语句),(3)变量 v 在语句 $S1$ 上使用出现。

定义 3(依赖链) 依赖链是两个以上的控制或数据依赖关系组成的依赖关系序列,且在该序列中,一个依赖关系的目标语句是下一个依赖关系的起始语句。

上述覆盖准则从代码修改处,根据控制依赖或数据依赖关系进行前向遍历以识别出受影响的程序实体,其本质是执行前向切片。但从 PIE 模型角度进行分析,发现该准则仅满足 E 条件,而未考虑到 I 条件和 P 条件;同时还存在如下不足:(1)测试需求在计算时所需的计算开销和存储开销均较大,所以方法的可扩展性较差,难以适用于大规模程序。(2)该覆盖准则对依赖链的长度未加以限定,其不足将直接导致最终生成的测试需求的数量过大。

针对上述不足,Santelices 等人结合依赖性分析(Dependence Analysis)和部分符号执行(Partial Symbolic Execution)[17]来设计测试覆盖准则[7]。该准则的设定基于如下两个观察:(1)设计时需要同时从控制依赖和数据依赖角度考虑代码修改可能影响的语句并确保测试用例可以覆盖到这些语句,所以该方法同时考虑了与代码修改相关的特定数据和控制依赖链。(2)测试覆盖准则的设定需要额外考虑程序的运行状态。

为了进行依赖性分析,该方法将被测程序用程序依赖图(Program Dependence Graph)[41]进行建模,其定义如下所述。

定义 4(程序依赖图) 该模型可同时描述语句间的控制依赖和数据依赖关系,其中节点表示语句(块),边表示语句(块)间的控制或数据依赖关系。

基于上述模型，在准则设定时，首先基于代码修改生成所有指定长度的依赖链集合。其次在给定的依赖链基础上，采用部分符号执行(Partial Symbolic Execution)来对代码修改影响进行分析，即符号执行从代码修改处进行分析，而不是从被测程序的入口处进行分析。对于同一个测试用例，分析时主要考虑两种情况：(1)该测试用例覆盖到依赖链终点，但未能覆盖到另一个程序对应的依赖链终点。(2)虽然能覆盖到同一依赖链，但在依赖链终点上的程序状态却不一致。根据上述分析，可以看出该测试准则的设定不仅满足了 PIE 模型的 E 条件，而且对 P 条件和 I 条件也进行了一定的考虑。

在他们的研究中，为了降低符号执行时的计算开销和生成的测试需求数量，对依赖链的长度同样进行了限定，但这影响了方法的执行效果。为了支持更长依赖链的分析，Santelices 等人在原有研究工作的基础上提出了一种新的方法[13]。该方法基于动态切片技术[42]，即不需要预先计算所有的测试需求，而是在测试用例执行时进行依赖性分析和状态差异检测，该方法在保持原有方法精度的基础上，可以有效避免符号执行的开销，从而有效支持更长依赖链的分析。

Person 等人对传统的符号执行法进行扩展，并提出一种更为通用的差异化符号执行法(Differential Symbolic Execution)[9]来对程序行为进行分析，最终识别出的测试需求精度要高于上述方法[7,17]。该方法可以支持代码重构、回归测试用例集维护等软件开发活动。针对测试用例集扩充问题，使用差异化符号执行可以生成一系列测试需求，并通过约束求解器对相应的约束进行联合求解来生成测试用例。

徐志弘等人提出有向测试用例集扩充技术(Directed Test Suite Augmentation)[8,14-16]，采用安全的测试用例选择技术(安全的测试用例选择技术可以确保未被选择的测试用例不可能检测到修改后程序内的缺陷)，即 DejaVu 方法[21]，将已有测试用例划分为两部分。具体来说，DejaVu 在修改前后程序对应的控制流图(Control Flow Graph)上同时执行深度优先遍历策略来确定与修改相关的危险边集，然后将已有测试用例集 T 划分为 T_a 和 T_u，其中 T_a 包含覆盖到危险边的测试用例，而 T_u 包含未覆盖到危险边的测试用例。在修改后程序 P' 上重新执行 T_a 中的所有测试用例并计算出覆盖的程序分支，同时结合 T_u 在修改前程序 P 上覆盖的程序分支，可以计算出已有测试用例在 P' 上尚未覆盖的程序分支。上述分支构成了目标测试需求并随后用于指导新测试用例的设计。

3.2 新测试用例生成算法

给定目标测试需求(例如程序语句、分支和路径)，一般借助手工方式来设计测试用例，但目前存在两类常见的自动测试用例生成方法(即动态符号执行法和演化测试法)，其得到学者的深入研究。

动态符号执行法(Dynamic Symbolic Execution 或 Concolic Testing)[44-46]最早由 Sen 等人提出，通过引入测试用例的具体执行，可以有效解决传统符号执行法的不足。其执行流程可以简要描述如下：首先随机生成一个测试用例，执行该测试用例并搜集对应执行轨迹上的路径条件(Path Condition)。接着对该路径条件上最后一个谓词执行取反操作并生成一个新的路径条件，如果该路径条件并未出现过，则调用约束求解器进行求解并生成新的测试用例，否则对路径条件上倒数第二个谓词进行取反操作。当不能生成新的路径条件时，该迭代过程将终止和结束。虽然动态符号执行法在理论上可以生成一系列测试用例来遍历被测程序内的所有可行路径，但在实际执行过程中却受到约束求解器求解能力的约束。

目前动态符号执行法已经成功应用于测试用例集扩充技术，除了用于指导与代码修改的测试覆盖准则的设定[7,13,17]，也可以用于指导新的测试用例的生成。

徐志弘等人提出一种结合测试用例选择和动态符号执行法的测试用例集扩充技术[16]。其将修改后程序尚未覆盖的分支作为目标测试需求，尝试采用动态符号执行技术设计新的测试用例来完成对目标测试需求的覆盖。

齐大伟等人针对单处代码修改，提出一种基于动态符号执行法的测试用例集扩充方法[12]。该方法可以总结为两个步骤：步骤 1，首先考虑满足 PIE 模型的 E 条件，即给定单处代码修改 c，在控制流图中搜索覆盖 c 的路径(该搜索过程可以借助控制依赖图来提高搜索效率)，随后通过动态符号执行法可以生成覆盖该路径的测试用例 t。步骤 2，则进一步考虑满足 PIE 模型的 P 条件和 I 条件，即在已有测试用例 t 上尝试重新生成新的测试用例，以避免程序内部状态出现异常而随后影响程序的最终输出。他们对代码修改影响程序输出的常见模式进行总结，其中需要避免的模式包括：(1)虽然对代码修改影响的变量进行定义，但随后并未使用过。(2)代码修改影响的变量虽然使用过，但并未对程序的最终输出造成影响。该方法也同时考虑了复用已有的测试用例。譬如在步骤 1 中若存在测试用例覆盖到代码修改，则直接返回该测试用例，否则通过搜集已有测试用例的路径条件来指导生成新的测试用例，以覆盖代码修改。

与齐大伟等人的研究工作相似，Taneja 等人提出回归测试用例生成(Regression Test Generation)概念，尝试直接生成可以检测出修改前后程序执行行为差异的测试用例(即同一测试用例，在修改前后程序上的实际输出并不一致。测试人员借助这种行为差异可以判断出代码修改是否符合他们的预期)[11]，其研究工作建立在动态符号执行法的基础之上[44-46]。他们发现传统的测试用例生成需要遍历被测程序的所有可能执行路径，而在回归测试用例生成时仅需考虑会导致执行行为差异的程序路径，从而提高测试用例集的扩充效率。该方法首先识别出 P' 中所有不满足 PIE 条件的无关分支，然后提出一种新的搜索策略来避免探索到这类无关分支。除此之外，还充分利用已有测试用例的覆盖信息来进一步提高搜索效率。该方法主要包括 4 个模块：(1)代码修改识别模块，该模块对修改前后程序进行比较得出代码修改操作序列。(2)图模型构造模块，该模块对修改后程序用控制流图进行建模并对与代码修改相关的节点进行标记。(3)图遍历模块，该模块对构造出的控制流图进行遍历，识别出无助于检测出执行行为差异的程序分支。这些分支可简要分为两类：B_{E+I} 和 B_P。其中覆盖 B_{E+I} 中的分支不能确保覆盖到代码修改处并导致程序内部状态异常，覆盖 B_P 中的分支不会导致程序内部异常的状态传播到程序输出上。(4)测试用例动态生成模块，该模块采用动态符号执行法，根据指定程序路径来设计新的测试用例，并避免对覆盖了无关分支的路径进行探测。同时充分利用已有测试用例来进一步提高搜索效率，即避免从空树开始搜索，而是利用已有测试用例来预先构造动态执行

树，并从该动态执行树上开始增量搜索。

演化测试法（典型算法包括爬山法、模拟退火、遗传算法和禁忌搜索等）同样可以用于指导新的测试用例的设计[47,48]。遗传算法（Genetic Algorithms）是一种全局搜索算法，它主要受到生物界进化规律（即适者生存和优胜劣汰机制）的启发。针对具体问题，遗传算法需要对候选解进行染色体编码和适应值函数设定。针对测试用例生成问题，候选测试用例可以直接编码为染色体，适应值函数的设定可以反映出染色体满足指定测试覆盖目标的情况。在生成单个测试用例时，从初始种群（一般由随机生成的多个测试用例构成）开始，不停进行种群演化，直至覆盖到目标测试需求或达到指定迭代次数为止。在每轮种群演化时，依次执行选择操作、交叉操作和变异操作可以形成新的种群。徐志弘等人采用遗传算法来指导测试用例集扩充[15]，他们识别出算法中的影响因素并通过实证研究对这些因素进行了深入分析。识别出的影响因素包括：(1)与代码修改相关的程序实体识别算法；(2)已有测试用例集特征，例如代码规模和代码覆盖率；(3)在测试用例生成时需要覆盖的程序实体的考虑次序；(4)遗传算法在每一次迭代时是否利用上一次迭代过程中生成的测试用例。

最后徐志弘等人将上述两项研究工作[15,16]总结到一个统一方法框架，同时进行了系统分析和比较[8]。主要分析了其中3个影响因素：(1)在生成测试用例时，需要被覆盖的程序实体的考虑次序。因为覆盖一个程序实体时可能会无意覆盖其他程序实体，所以合理的遍历策略可以有效提高测试用例集扩充方法的效果。(2)已有测试用例和新生成测试用例的使用方式。(3)采用的测试用例生成方法，这里主要考虑的是文献[15]中采用的演化测试法和文献[16]中采用的动态符号执行法。最终的实证研究结果表明：影响有向测试用例集扩充方法有效性和时间开销的最重要因素是测试用例生成算法；其次是已有或新生成测试用例的使用方式，该因素对时间开销影响较大，但对有效性影响并不明显。受影响程序实体的考虑次序对演化测试法效果的影响要大于动态符号执行法。

在上述两种测试用例生成算法的基础上，徐志弘等人进一步推断，若将上述两种测试用例生成方法进行混合，可有效提高测试用例的生成效率。他们提出一种混合测试用例生成方法[14]。该方法分多个阶段，在每一个阶段，首先采用动态符号执行法，然后使用演化测试法。在每一个阶段开始时，方法的输入是一系列需要覆盖的分支集合和上一阶段输出的一系列已有测试用例，其中分支按照深度优先策略进行排序。

3.3 研究工作的实现

这一小节主要考察已有测试用例集扩充工具的实现细节，包括功能模块构成和各个模块开发中涉及的一些工具。

Apiwattanapong等人首先开发出MATRIX工具[17]以高效支持回归测试，该工具借助Jdiff[49]工具来获得代码修改信息（考虑的代码修改操作包括增加语句操作、删除语句操作和修改语句操作）。MATRIX工具主要由3大模块构成。其中IDENTIFIER模块识别与代码修改相关的测试需求，该模块基于JABA框架[50]并使用一个基于JABA的程序切片工具来计算数据和控制依赖链。INSTRUMENTER模块对修改后代码进行插桩以便搜集测试用例的测试需求覆盖情况。ANALYZER模块对记录的信息进行分析便可确定哪些测试需求已经被测试用例覆盖到。后两个模块均基于Eclipse插件项目INSECTJ[51]。随后Santelices等人在MATRIX工具基础上进一步开发了MATRIXRELOAD工具集[7]，该工具集采用Soot分析框架[52]对Java字节码文件进行了分析和插桩。MATRIXRELOAD工具集主要由两个模块构成。其中程序分析模块主要生成与代码修改相关的测试需求，该模块包括依赖分析器和符号执行引擎。程序运行监控模块主要对修改后程序进行插桩并记录测试用例在运行时的测试需求覆盖信息。

徐志弘等人提出有向测试用例集扩充技术[8,14-16]，他们采用Sofya分析系统[53]完成代码插桩、控制流图构造和Dejavu算法的实现。采用Soot框架[52]来搜集测试用例执行轨迹上的路径条件。当给定目标测试需求，编码实现动态符号执行法和演化测试法来设计新的测试用例。

齐大伟等人在BitBlaze二进制代码分析框架[54]基础上实现了他们提出的测试用例集扩充法[12]，为了实现步骤1，首先采用ERESI工具来生成控制流图，使用CDG builder模块来构造程序依赖图；其次使用TEMU模块来搜集测试用例在修改后程序上的执行路径；随后采用VINE模块来生成执行路径上的路径条件。为了实现步骤2，基于ETMU模块构造修改影响传播树，并与CDGbuilder模块协同构造出可以检测出执行行为差异的测试用例。

Taneja等人在Pex工具[46]基础上进行增量开发并提供了eXpress工具来支持测试用例集扩充[11]。而Person等人基于Java的PathFinder[55]分析工具集实现了他们提出的差异化符号执行法[9]。

通过上述分析，可以看出在测试用例集扩充技术研究中存在大量可以使用的优秀程序分析和测试工具，例如：Jdiff、Soot、BitBlaze、Pex和PathFinder等。熟悉和利用上述工具可以快速实现自己的解决方案，从而提高研究效率。

3.4 已有研究工作系统比较

表2 对已有研究工作的系统比较

文献	发表年份	建模方法	程序实体	测试需求生成方法	新的测试用例生成方法	复用已有测试用例	支持多处代码修改
Binkley[18]	1997	系统依赖图	程序路径	方法1	N/A	否	否
Rothermel等[10]	1994	系统依赖图	程序路径	方法1	N/A	否	否
Santelices等[7,13,17]	2006、2008、2011	程序依赖图	程序路径	方法2	N/A	否	是
徐志弘等[8,16]	2009、2010	控制流图	语句分支	方法3	动态符号执行法	是	否
徐志弘等[8,15]	2010	控制流图	语句分支	方法3	演化测试法	是	否
徐志弘等[14]	2011	控制流图	语句分支	方法3	混合法	是	否
Person等[9]	2008	源代码	程序语句	方法4	符号执行	否	否
齐大伟等[12]	2010	控制流图	程序路径	方法5	动态符号执行法	是	否
Taneja等[11]	2011	控制流图	程序路径	方法5	动态符号执行法	是	否

这一小节对已有研究工作进行系统比较，通过上述分析可知，识别出的主要比较因素包括文献发表年份、被测程序建模方法、程序实体、测试需求生成方法、新的测试用例生成方法、是否复用已有测试用例以及是否支持多处代码修改等。其中测试需求常见生成方法总结如下：方法 1，基于切片技术；方法 2，基于控制和数据依赖关系并同时考虑程序运行状态；方法 3，通过在修改后程序上运行已有测试用来确定尚未覆盖的语句分支；方法 4，采用差异化符号执行方法；方法 5，基于 PIE 模型生成测试需求。最终的比较结果如表 2 所列，表中 N/A 表示并没有指定具体测试用例生成方法。

4 评测数据集

在测试用例集扩充技术研究中，研究人员为了评估所提技术的优越性，经常使用西门子套件(Siemens Suite)中的程序作为标准评测数据。西门子套件最早是为了研究控制流和数据流准则对缺陷检测能力的影响而创建的，主要包含了 7 个 C 语言程序。这些程序代码规模较小，其代码行数一般介于 200 行到 500 行之间。程序中的缺陷注入由西门子研究人员完成并确保与实际缺陷保持一致。这些程序可以从内布拉斯加大学林肯分校的 SIR 库中下载[56]。例如 Santelices 等人在文献[17]中选择了 tcas 和 schedule 程序，在文献[7]中选择了 tcas 程序，在文献[13]中选择了 totinfo、schedule、schedule2和`printtok。但他们采用 Soot 等工具对程序进行分析，所以将从西门子套件中选择的程序转换成 Java 语言版本，例如转换后的 tcas 程序包括 2 个类、10 个方法和 134 行非注释代码，schedule 程序包括 1 个类、18 个方法和 268 行非注释代码。徐志弘等人在文献[16]中选择了 tcas 程序，在文献[8]中选择了 printtok、printtok2、replace 和 tcas 程序，在文献[14]中选择了 printtok、printtok2 和 replace 程序。除了文献[16]采用的是 Java 语言版本，文献[8,14]采用的均是 C 语言版本。齐大伟等人在文献[12]中选择了 tcas 程序。Taneja 等人在文献[11]选择了 replace 程序。但他们开发的工具基于 Pex[46]，所以将该程序转换为 C# 语言版本。

除了上述小规模程序，研究人员为了突出所提方法的扩展性，也使用了一些中等规模和大规模程序。例如 Santelices 在文献[7,13]中选择了 NanoXML 程序，该程序主要负责 XML 文件的解析，NanoXML 是一个典型的大规模程序并具有面向对象程序的一些基本特征，例如封装、继承、多态和异常处理等。在他们的实证研究中对版本 1 考虑了 7 个修改版本，对版本 5 则仅考虑了 2 个修改版本。同样，徐志弘等人在文献[15]中也选择了 NanoXML 程序，通过该实验对象对测试用例集扩充技术中的影响因素进行了分析。齐大伟等人在文献[12]中选择了 LibPNG 程序，该程序包含大约 28000 行代码，主要用于操作 PNG 图像。实证研究表明，针对不同程序版本，他们所提方法均可以生成在修改前后程序上执行行为存在差异的测试用例。Taneja 等人在文献[11]选择了 STPG、Siena 和 Structorian 程序。其中 STPG 是 codeplex 网站管理的开源软件[57]，包含大约 684 行代码。Siena 来自 Sir 库[56]，包含大约 1529 行代码。Structorian 程序是 Google 管理的一个开源项目[58]，主要用于支持二进制数据的查阅和逆向工程，包含大约 6561 行代码。由于上述程序均是 Java 语言版本，为了评估方法的有效性，他们采用 java 2 csharp translator 工具[59]将这些程序转化成 C# 语言版本。Person 等人在文献[9]中将差异化符号执行法应用到测试用例集扩充问题上，并采用 Siena 程序作为实验对象。

最后表 3 从程序名称、程序简介、编程语言、使用的文献、代码规模和总共使用的次数等方面对所有的评测数据进行了系统总结。

表 3 评测数据总结

程序名称	程序简介	编程语言	使用的文献	代码规模	总共使用次数
printtok	词法分析程序	C/Java	[8,13,14]	小规模	3
printtok2	词法分析程序	C	[8,14]	小规模	2
replace	模式识别	C/C#	[8,11,14]	小规模	3
schedule	优先调度程序	Java	[13,17]	小规模	2
schedule2	优先调度程序	Java	[13]	小规模	1
tcas	航空交通冲突检测算法	C/Java	[7,8,12,16,17]	小规模	5
totinfo	信息度量程序	Java	[13]	小规模	1
STPG	codeplex 网站主持的开源程序	C#	[11]	中等规模	1
NanoXML	XML 解析器	Java	[7,13,15]	大规模	3
LibPNG	用于处理 PNG 图像文件的开源库	C	[12]	大规模	1
Siena	Internet-scale 事件通知程序	Java/C#	[9,11]	中等规模	2
Structorian	支持二进制数据查阅和逆向工程	C#	[11]	大规模	1

5 评测指标

目前对测试用例集扩充技术有效性的评测指标主要分两个方面：有效性和执行效率。

有效性评测指标：Santelices 等人针对基于代码修改的测试覆盖准则提出一种评测指标[7,13,17]。给定一个测试覆盖准则，假设存在一个满足该测试覆盖准则的测试用例集 T，其中可以检测出修改前后程序执行行为差异的测试用例构成集合 T_d，则该评测指标 ds 可通过如下公式进行度量：

$$ds=|T_d|/|T| \tag{1}$$

ds 指标可以评测出该覆盖准则针对代码修改的缺陷检测能力，其取值越大，则代表检测出缺陷的概率越高。

徐志弘等人在文献[8,14-16]中考虑了另一种有效性评测指标，即考察扩充后的测试用例集的程序分支的覆盖率。分支覆盖率越高，则测试用例集扩充技术也越有效。齐大伟等人则在给定代码修改后，以是否能够生成检测出修改前后程序行为差异的测试用例作为方法有效性的评测指标[12]。

执行效率指标：研究人员一般考察测试用例集扩充技术的执行时间，一种简单方法是统计物理执行时间，例如徐志弘等人在文献[8,14]和 Taneja 等人在文献[11]中均采用了该方法。同样，研究人员也有从统计重要操作调用次数的角度来评测执行效率。例如徐志弘等人在文献[16]中将动态符号执行法对约束求解器的调用次数进行了统计。Taneja 等人在文献中也对动态符号执行的次数进行了统计[11]。同样，徐志弘等人在文献[15]中将演化测试法对测试用例的执行次数进行了统计。

6 测试用例集扩充问题研究面临的挑战

回归测试中的测试用例集扩充问题作为近几年的研究热点在未来仍然存在很多研究点值得进一步关注，这一节对一

些研究点进行尝试性探讨。

(1)充分利用目前在代码修改影响分析[43,60]领域的最新研究成果,精确高效地识别出与代码修改存在依赖关系的其他程序实体,从而提高测试用例集扩充技术的有效性。同时对于识别出的这些测试需求,设计出更为高效的测试用例自动生成方法。目前一种可行的思路是借鉴混合动态符号执行法和演化测试法的测试用例生成技术[61,62]来提高测试用例集扩充效率。

(2)目前大部分研究工作假设代码修改操作较为简单,即仅针对被测程序的单处代码修改进行测试用例集扩充。但在实际的测试场景中,开发人员可能会针对被测程序一次修改多处代码。当多处代码修改之间存在依赖关系时,将会进一步增加代码修改影响分析的难度。

(3)进一步识别出影响测试用例集扩充技术有效性的影响因素,通过更多的实证研究去分析这些因素的实际影响效果。这些可能影响因素包括被测程序的特征、代码修改类型、测试用例集特征。例如针对测试用例集特征,可以考虑如下属性:测试用例集规模、测试用例输入是否呈多样化和测试用例的具体覆盖率等。以测试用例覆盖率为例,如果已有测试用例集的代码覆盖率较高,则仅需花费较小的代价就可以完成测试用例集扩充。

(4)目前研究工作考虑的大部分实验对象代码规模还较小,需要在下一步工作中进一步提高已有研究工作的扩展性并开发出相应的工具集,使其可以更好地用于指导实际软件企业的大规模面向对象程序的回归测试。其中一种研究思路是采用软件工程中的 UML 模型对被测程序进行建模,并以该模型为基础提出相应的测试用例集扩充方法。这一研究点可以参考文献[63-65]的研究工作。

(5)缺陷定位是一种有效的自动软件调试技术,已有研究成果表明测试用例的维护(例如测试用例集约简和测试用例优先级排序等)对缺陷定位效果存在重要影响[66-68]。随着测试用例集扩充技术的深入研究,进一步探讨旨在提高缺陷定位效果的测试用例集扩充技术值得研究人员关注。

(6)将测试用例扩充的已有研究成果与企业的实际测试流程进行结合。论文中提出的方法一般采用对照试验方式进行有效性评估,所得结论难以适用于软件企业的大型软件产品。其次已有的工具未考虑到与软件企业目前已有的开发、测试和调试流程相结合。若要让企业接受高校的研究成果,需要进一步提高测试工具的自动化程度、改善工具的用户界面并保证工具具有一定的鲁棒性。若对上述挑战提出有效解决方案,将能有效提高研究成果的应用价值并大幅度提高企业内的测试和调试效率。一种解决方案是采用 Eclipse 插件方式进行开发。

结束语 测试用例集扩充问题是回归测试研究中的最新研究热点。在代码修改影响分析基础上,对已有测试用例集的充分性进行评估,若不充分则通过设计新的测试用例来完成对代码修改的充分测试。本文首次对该研究问题的已有研究成果进行了系统的总结和比较。目前该研究领域仍很活跃,本文也对未来的可能研究方向进行了一些尝试性的探讨。相信针对这些问题提出的有效解决方案将进一步提高软件回归测试的效果并大幅度降低软件开发和维护费用。

参 考 文 献

[1] Beizer B. Software Testing Techniques [M]. New York: Van Nostrand Reinhold, 1990

[2] Leung H, White L. Insights into Regression Testing [C]// Proceedings of the International Conference on Software Maintenance. 1989: 60-69

[3] Harrold M J, Orso A. Retesting Software during Development and Maintenance [C] // Proceedings of Frontiers of Software Maintenance. 2008: 99-108

[4] Yoo S, Harman M. Regression Testing Minimization, Selection and Prioritization: a Survey [J]. Software Testing, Verification & Reliability, 2012, 22(2): 67-120

[5] 章晓芳,陈林,徐宝文,等. 测试用例集约简问题研究及其进展[J]. 计算机科学与探索,2008(3):235-247

[6] 屈波,聂长海,徐宝文. 回归测试中测试用例优先级技术研究综述[J]. 计算机科学与探索,2009(3):225-233

[7] Santelices R, Chittimalli PK, Apiwattanapong, et al. Test-suite Augmentation for Evolving Software [C]// Proceedings of the International Conference on Automated Software Engineering. 2008: 218-227

[8] Xu Z, Kim Y, Kim M, et al. Directed Test Suite Augmentation: Techniques and Tradeoffs [C]// Proceedings of the International Symposium on Foundations of Software Engineering. 2010: 257-266

[9] Person S, Dwyer M B, Elbaum S, et al. Differential Symbolic Execution [C]// Proceedings of the International Symposium on Foundations of Software Engineering. 2008: 226-237

[10] Rothermel G, Harrold M J. Selecting Tests and Identifying Test Coverage Requirements for Modified Software [C]// Proceedings of the International Symposium on Software Testing and Analysis. 1994: 169-184

[11] Taneja K, Xie T, Tillmann N, et al. eXpress: Guided Path Exploration for Regression Test Generation [C]// Proceedings of the International Symposium on Software Testing and Analysis. 2011: 1-11

[12] Qi D, Roychoudhury A, Liang Z. Test Generation to Expose Changes in Evolving Programs [C]// Proceedings of the International Conference on Automated Software Engineering. 2010: 397-406

[13] Santelices R, Harrold M J. Applying Aggressive Propagation-based Strategies for Testing Changes [C]// Proceedings of the International Conference on Software Testing. 2011: 11-20

[14] Xu Z H, Kim Y, Kim M, et al. A Hybrid Directed Test Suite Augmentation Technique [C]// Proceedings of the International Symposium on Software Reliability Engineering. 2011: 150-159

[15] Xu Z, Cohen M B, Rothermel G. Factors Affecting the Use of Genetic Algorithms in Test Suite Augmentation [C]// Proceedings of the Annual Conference on Genetic and Evolutionary Computation. 2010: 1365-1372

[16] Xu Z, Rothermel G. Directed Test Suite Augmentation [C]// Proceedings of the Asia-Pacific Software Engineering Conference. 2009: 406-413

[17] Apiwattanapong T, Santelices R, Chittimalli PK, et al. Matrix: Maintenance-oriented Testing Requirement Identifier and Examiner [C]// Proceedings of the Testing: Academic and Industrial Conference-Practice and Research Techniques. 2006: 137-146

[18] Binkley D. Semantics Guided Regression Test Cost Reduction

[J]. IEEE Transactions on Software Engineering, 1997, 23: 498-516

[19] Daniel B, Jagannath V, Dig D, et al. Reassert: Suggesting Repair for Broken Unit Tests [C]// Proceedings of the International Conference on Automated Software Engineering. 2009: 433-444

[20] Daniel B, Gvero T, Marinov D. On Test Repair using Symbolic Execution [C]// Proceedings of the International Symposium on Software Testing and Analysis. 2010: 207-218

[21] Rothermel G, Harrold M. A Safe, Efficient Regression Test Selection Technique [J]. ACM Transactions on Software Engineering and Methodology, 1997, 6(2): 173-210

[22] Rothermel G, Harrold M. Empirical Studies of a Safe Regression Test Selection Technique [J]. IEEE Transactions on Software Engineering, 1998, 24(6): 401-419

[23] Rothermel G, Harrold M, Dedhia J. Regression Test Selection for C++ Software [J]. Software Testing, Verification and Reliability, 2000, 10(2): 77-109

[24] Harrold M, Gupta R, Soffa M. A Methodology for Controlling the Size of a Test Suite [J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285

[25] Chen T, Lau M. A New Heuristic for Test Suite Reduction [J]. Information and Software Technology, 1998, 40(5/6): 347-354

[26] Hsu H, Orso A. MINTS: A General Framework and Tool for Supporting Test-suite Minimization [C]// Proceedings of the International Conference on Software Engineering. 2009: 419-429

[27] 章晓芳,徐宝文,聂长海,等. 一种基于测试需求约简的测试用例集优化方法[J]. 软件学报, 2007, 18(4): 821-831

[28] 顾庆,唐宝,陈道蓄. 一种面向测试需求部分覆盖的测试用例集约简技术[J]. 计算机学报, 2011, 34(5): 879-888

[29] Chen X, Zhang L, Gu Q, et al. A Test Suite Reduction Approach based on Pairwise Interaction of Requirements [C]// Proceedings of the ACM Symposium on Applied Computing. 2011: 1395-1402

[30] Zhang L, Chen X, Gu Q, et al. CATESR: Change-aware Test Suite Reduction Based on Partial Coverage of Test Requirements [C]// Proceedings of the International Conference on Software Engineering and Knowledge Engineering. 2012: 217-224

[31] Rothermel G, Untch R, Chu C. Prioritizing Test Cases for Regression Testing [J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948

[32] Elbaum S, Malishevsky A, Rothermel G. Test Case Prioritization: A Family of Empirical Studies [J]. IEEE Transactions on Software Engineering, 2002, 28(2): 159-182

[33] Li Z, Harman M, Hierons R. Search Algorithms for Regression Test Case Prioritization [J]. IEEE Transactions on Software Engineering, 2007, 33(4): 225-237

[34] Zhang L, Hou S, Guo C, et al. Time-Aware Test-Case Prioritization using Integer Linear Programming [C]// Proceedings of the International Symposium on Software Testing and Analysis. 2009: 213-224

[35] Do H, Mirarab S, Tahvildari L, et al. The Effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments [J]. IEEE Transactions on Software Engineering, 2010, 36(5): 593-617

[36] Do H, Rothermel G. An Empirical Study of Regression Testing Techniques incorporating Context and Lifetime Factors and Improved Cost-benefit Models [C]// Proceedings of the Symposium on the Foundations of Software Engineering. 2006: 141-151

[37] Malishevsky A G, Rothermel G, Elbaum S. Modeling the Cost-benefits Tradeoffs for Regression Testing Techniques [C]// Proceedings of the International Conference on Software Maintenance. 2002: 204-213

[38] Do H, Rothermel G. Using Sensitivity Analysis to Create Simplified Economic Models for Regression Testing [C]// Proceedings of the International Symposium on Software Testing and Analysis. 2008: 51-62

[39] Voas J. PIE: A Dynamic Failure-based Technique [J]. IEEE Transactions on Software Engineering, 1992, 18(8): 352-357

[40] Zhu H, Hall P, May J. Software Unit Test Coverage and Adequacy [J]. ACM Computing Survey, 1997, 29(4): 366-427

[41] Ferrante J, Ottenstein K, Warren J. The Program Dependence Graph and its Use in Optimization [J]. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319-349

[42] Weiser M. Program Slicing [J]. IEEE Transactions on Software Engineering, 1984, 10(4): 352-357

[43] Bohner S, Arnold R. Software Change Impact Analysis [M]. Los Alamitos, 1996

[44] Sen K, Marinov D, Agha G. CUTE: a Concolic Unit Testing Engine for C [C]// Proceedings of the International Symposium on Foundations of Software Engineering. 2005: 263-272

[45] Godefroid P, Klarlund N, Sen K. DART: Directed Automated Random Testing [C]// Proceedings of the Conference on Programming Language Design and Implementation. 2005: 213-223

[46] Tillmann N, Halleux J. Pex-white Box Test Generation for .NET [C]// Proceedings of the International Conference on Tests and Proofs. 2008: 134-153

[47] McMinn P. Search-based Software Test Data Generation: a Survey [J]. Software Testing, Verification and Reliability, 2004, 14(2): 105-156

[48] 陈翔,顾庆,王子元,等. 一种基于粒子群优化的成对组合测试算法框架 [J]. 软件学报, 2011, 22(12): 2879-2893

[49] Apiwattanapong T, Orso A, Harrold M J. A Differencing Algorithm for Object-oriented Programs [C]// Proceedings of the International Conference on Automated Software Engineering. 2004: 2-13

[50] http://www.cc.gatech.edu/aristotle/Tools/jaba.html

[51] Seesing A, Orso A. InsECTJ: A Generic Instrumentation Framework for Collecting Dynamic Information within Eclipse [C]// Proceedings of the eclipse Technology eXchange Workshop at OOPSLA. 2005: 49-53

[52] http://www.sable.mcgill.ca/soot

[53] http://sofya.unl.edu

[54] http://bitblaze.cs.berkeley.edu

[55] http://babelfish.arc.nasa.gov/trac/jpf

[56] Do H, Elbaum S, Rothermel G. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact [J]. Empirical Software Engineering, 2005, 10(4): 405-435

[57] http://stringtopathgeometry.codeplex.com

[58] http://code.google.com/p/structorian

[59] http://sourceforge.net/projects/j2cstranslator

[60] Sun X, Li B, Zhang S, et al. Using Lattice of Class and Method Dependence for Change Impact Analysis of Object Oriented Programs [C]// Proceedings of the Symposium on Applied Computing. 2011: 1444-1449

(下转第 28 页)

的关系，决定了估算必须以过程的方式来组织。估算和开发管理彼此协调，相互促进，一起形成了一个更加完整的有机整体，提高了软件项目成功率。

本文根据软件开发与管理过程对估算活动的需要，同时考虑到估算为更好地支持开发与管理要解决的关键问题，给出了估算过程模型，它包括两部分，一是 RUP 估算过程，二是应用贝叶斯网络对估算过程建立的推理模型。简言之，主要工作在于：

1. 估算和开发管理一样，是软件项目过程中展开的 3 个方面的活动，三者是密不可分、相互促进的。不能孤立地研究估算方法来解决估算相关的全部问题。通过估算过程模型方案，解决了开发与管理中缺乏估算活动定义的问题，同时也为估算本身急需解决的 3 个问题提供了可行途径。

2. 详细的步骤式估算过程，为干系人提供了清晰的估算线路图。

3. 贝叶斯网络推理模型以图形化的方式，结合历史数据、专家经验和项目本身的数据，进行分析、权衡和评价。

本文将得到估算结果的 RUP 估算过程模型与进行分析权衡的贝叶斯网络模型独立开来，一方面可获得详细的估算指南，另一方面可获得清晰的分析视图，在不失细节的情况下，保持了贝叶斯网络模型的简单性，避免了在贝叶斯网络模型中引入不必要的不确定性，但又不会使模型局限于特定的项目范围。

案例分析验证了本估算过程模型具有良好的适用性，对项目开发与管理有极大的推动作用。

参 考 文 献

[1] Brooks F P. The Mythical Man-Month[M]. Addison-Wesley Professional, 1995

[2] Boehm B W. Software Engineering Economics [M]. Prentice Hall PTR, 1981

[3] Boehm B W, Abts C, Brown A W, et al. Software Cost Estimation with COCOMO II[M]. Prentice Hall PTR, 2000

[4] Jones C, Engineering S. The state of the Art in 2005(Version 5) [M]. Software Productivity Research WhitePaper, February 2005

[5] Jrgensen M, Shepperd M. A systematic review of software development cost estimation studies[J]. IEEE Transactions on Software Engineering, 2007, 33(1): 33-53

[6] Stutzke R D. Estimating Software-Intensive Systems[M]. Upper Saddle River, NJ: Addison-Wesley, 2005

[7] Goldratt, Eliyahu M. Critical Chain[M]. MA: The North River Press, 1997

[8] Jones C. Software Assessments. Benchmarks, and Best Practices, Reading[M]. MA: Addison-Wesley, 2000

[9] Project Management Institute. A guide to the project management body of knowledge (PMBOK Guide) (Fourth Edition) [M]. ANSI/PMI 99-001-2008

[10] Shepperd M. Software project economics: a roadmap[C]// Future of Software Engineering(FOSE'07). IEEE, 2007

[11] Yang Da, Wang Qing, Li Ming-shu, et al. A Survey on Software Cost Estimation in the Chinese Software Industry[C]// ESEM'08. Kaiserslautern, Germany, October 2008

[12] Fraser S, Jrgensen M, Boehm B, et al. The role of judgment in software estimation[C]// 31st International Conference on Software Engineering. Companion Volume, ICSE 2009. 2009: 13-17

[13] Ahmed F, Bouktif S, Serhani A, et al. Integrating function point project information for improving the accuracy of effort estimation[C]// The 2nd International Conference on Advanced Engineering Computing and Applications in Sciences, ADVCOMP 2008. 2008: 193-198

[14] Wang Hao, Peng Fei, Zhang Chao, et al. Software project level estimation model framework based on Bayesian belief networks [C]// International Conference on Quality Software. 2006: 209-216

[15] Yang Y, Jesal B, Boehm B. Value-based processes for COTS-based applications[J]. IEEE Software, 2005, 22(4): 54-62

[16] Jones C. Patterns of Software Systems Failure and Success[M]. International Thomson Press, 1996

[17] AgenaRisk 5.0 User Manual[OL]. www.agenarisk.com

[18] Steve McConnell. 软件估算—"黑匣子"揭秘[M]. 北京：电子工业出版社，2007

[19] Jones C. 软件项目估计[M]. 北京：电子工业出版社，2008

[20] Jacobson I, Booch G, Rumbaugh J. 统一软件开发过程[M]. 北京：机械工业出版社，2002

[21] 李明树，何梅，杨达，等. 软件成本估算方法及应用[J]. 软件学报，2007，18(4)：775-795

(上接第 15 页)

[61] Inkumsah K, Xie T. Improving Structural Testing of Object-Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution [C]// Proceedings of the International Confe-rence on Automated Software Engineering. 2008: 297-306

[62] Baars A, Harman M, Hassoun Y, et al. Symbolic Search-based Testing [C]// Proceedings of the International Conference on Automated Software Engineering. 2011: 53-62

[63] Briand LC, Labiche Y, Buist K, et al. Automating Impact Analysis and Regression Test Selection based on UML Designs [C]// Proceedings of the International Conference on Software Maintenance. 2002: 252-261

[64] Briand LC, Labiche Y, He S. Automating Regression Test Selection based on UML Designs [J]. Journal of Information and Software Technology, 2009, 51(1): 16-30

[65] Wu Y, Offutt J. Maintaining Evolving Component-based Software with UML [C]// Proceedings of the European Conference on Software Maintenance and Reengineering. 2003: 133-142

[66] Yu Y, Jones J, Harrold M. An Empirical Study of the Effects of Test-suite Reduction on Fault Localization [C]// Proceedings of the International Conference on Software Engineering. 2008: 201-210

[67] Artzi S, Dolby J, Tip F, et al. Directed Test Generation for Effective Fault Localization [C]// Proceedings of the International Symposium on Software Testing and Analysis. 2010: 49-60

[68] Zhang X, Gu Q, Chen X, et al. A Study of Relative Redundancy in Test-suite Reduction while Retaining or Improving Fault Localization Effectiveness [C]// Proceedings of the Symposium on Applied Computing. 2010: 2229-2236