软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2015,26(12):3043−3061 [doi: 10.13328/j.cnki.jos.004876]

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

# 构件软件的回归测试复杂性度量*

陶传奇[1,2,4], 李必信[3,4], Jerry Gao[5]

[1](南京理工大学 计算机科学与工程学院,江苏 南京 210094)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[3](东南大学 计算机科学与工程学院,江苏 南京 211189)

[4](东南大学 软件工程研究所,江苏 南京 211189)

[5](Department of Computer Engineering, San Jose State University, San Jose, CA, USA)

通讯作者: 陶传奇, E-mail: taochuanqi@njust.edu.cn

**摘 要**: 基于构件的软件构建方法目前被广泛使用在软件开发中,用于减少软件开发的工程成本和加快软件开发进度.在软件维护过程中,由于构件更新或者新版本的发布,基于构件的系统会受到影响,需要进行回归测试.对于指定的软件修改需求,维护者可以实施不同的修改手段.不同的修改手段会导致不同的回归测试复杂性,这种复杂性是软件维护成本和有效性的重要因素.目前的研究没有强调构件软件的回归测试复杂性问题.基于修改影响复杂性模型和度量,提出一种回归测试的复杂性度量框架.该度量框架包括两个部分:基于图的模型和形式化度量计算.该度量可以有效表示构件软件分别在构件和系统层面的回归测试复杂性因素,可视化地体现复杂性变化.然后根据模型,提出具体的度量计算方式.最后,通过实验研究,针对同一个构件软件的相同修改需求,利用若干个实验组进行独立修改实施,然后比较回归测试的复杂性.实验结果表明,所提出的度量方式是可行和有效的.

**关键词**: 基于构件的软件;回归测试;重测复杂性;软件维护

**中图法分类号**: TP311

中文引用格式: 陶传奇,李必信,Jerry Gao.构件软件的回归测试复杂性度量.软件学报,2015,26(12):3043−3061. http://www.jos.org.cn/1000-9825/4876.htm

英文引用格式: Tao CQ, Li BX, Gao J. Complexity metrics for regression testing of component-based software. Ruan Jian Xue Bao/Journal of Software, 2015,26(12):3043−3061 (in Chinese). http://www.jos.org.cn/1000-9825/4876.htm

## Complexity Metrics for Regression Testing of Component-Based Software

TAO Chuan-Qi[1,2,4], LI Bi-Xin[3,4], Jerry Gao[5]

[1](School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[3](School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)

[4](Institute of Software Engineering, Southeast University, Nanjing 211189, China)

[5](Department of Computer Engineering, San Jose State University, San Jose, CA, USA)

**Abstract**: Component-based software construction is a widely used approach in software development, aiming to reduce the engineering effort and speed up development cycle. During software maintenance, various software update approaches can be utilized to realize

* 基金项目: 国家自然科学基金(61402229, 61202003); 国家教育部博士点基金(20113219120021); 江苏省博士后基金(1401043B); 计算机软件新技术国家重点实验室(南京大学)开放课题(KFKT2015B10)

Foundation item: National Natural Science Foundation of China (61402229, 61202003); The Ph.D. Programs Foundation of Ministry of Education of China (20113219120021); the Postdoctoral Fund of Jiangsu Province (1401043B); The Open Fund of the State Key Laboratory for Novel Software Technology (Nanjing University) (KFKT2015B10)

收稿时间: 2014-12-20; 修改时间: 2015-03-09, 2015-06-11; 定稿时间: 2015-07-30

specific change requirements of component-based software. Different update approaches might lead to diverse regression testing complexity. However, there is a lack of research work addressing regression testing complexity in software maintenance. In this paper, a framework is proposed to measure and analyze regression testing complexity based on a set of change and impact complexity models and metrics. The paper presents an approach to complexity metrics for regression testing of component-based software. A graphic model and several measurements for the complexity metrics, which consist of both maintenance and retesting complexity, are also proposed. An experimental study is conducted to compare the complexity of regression testing using the data from several independent groups. The study results indicate the presented approach is feasible in providing visual comparison on various complexity of regression testing from different methods.

**Key words**: component-based software; regression testing; re-testing complexity; software maintenance

软件演化是软件的固有属性.在现代软件系统的生命周期内,系统需求改变、功能实现增强、新功能加入、软件架构改变、软件缺陷修复、运行环境改变均要求软件系统能够快速适应变化,具有较强的演化能力,从而要求软件工程人员能够快速适应改变,减少软件维护的代价.软件演化的研究与导致软件系统本身不一致的变化及其带来的影响效应相关,因而软件演化不仅是修改本身,还包括修改所带来的影响.这种影响可以是受到修改影响的文档规约、代码,也可以是受到影响的测试用例.因此,维护人员需要去理解、分析以及测试这些修改及其影响,以保证修改软件的一致性和可信性.回归测试是软件演化和软件维护中的必要内容之一,也是保证软件质量的重要手段.基于构件的软件工程(CBSE)已经被广泛应用在软件构造中,面向构件的系统主要由第三方提供的可重用构件或者内建的可重用构件组成.在软件维护过程中,当构件更新或者升级后,构件需要根据更新需求来进行回归测试,并且重新集成到构件应用系统中[1−3].

在前期工作中,我们初步解决了构件软件的系统化修改影响分析以及重测问题[4,5],定义了一些针对构件和系统的模型来支持构件软件修改影响分析,提出了一种系统化的从构件到整个软件系统的修改影响分析手段;然后,根据修改影响分析结果对测试用例更新,进行系统化的回归测试;最后,通过实际的构件系统进行实验验证,表明了方法的可行性和有效性.

在软件维护过程中,针对同一个修改需求,构件开发者或维护者可以采用不同的修改手段、修改类型、修改策略以及测试方法.这些都会给构件和构件所组成的系统带来不同程度的影响,从而导致不同的重测复杂性和回归测试成本.这种复杂性是软件维护有效性的重要因素之一,目前的研究还没有明确强调构件软件的回归测试复杂性.所以,如何有效度量、评估、预测回归测试复杂性和成本,已经成为测试管理员和软件质量保证工程师的一项重要任务.另外,复杂性度量也可以为成本预测提供有用的参考信息,对于提高软件维护的成本有效性有着重要意义.从软件维护角度来说,回归测试复杂性可以分为两部分:维护复杂性和重测复杂性.维护复杂性主要是由于修改和影响所导致,重测复杂性则与测试用例更新、测试用例执行、测试结果检查、重测环境等相关.因此,我们从整个回归测试过程出发,包括修改、影响以及测试用例更新 3 个方面,进行复杂性因素分析和提取.基于修改影响复杂性模型和度量,本文提出一种回归测试的复杂性度量框架,包括基于图形的模型和形式化的度量计算.在实验分析中,针对同一个构件软件的相同修改需求,利用若干个实验组进行独立修改实施,然后比较回归测试的复杂性,验证度量框架的可行性和有效性.

本文第 1 节对回归测试的复杂性度量因素进行分类和讨论.第 2 节根据这些复杂性因素提出本文的度量框架,具体介绍回归测试复杂性的度量过程.第 3 节是本文方法的实验研究和分析.第 4 节介绍当前软件回归测试复杂性的相关工作.第 5 节是本文结论和未来工作展望.

## 1 回归测试复杂性度量因素的分类讨论

研究表明:复杂性可以用来估算软件设计、编码、测试和维护的成本,同时也可以预测软件测试期间可能遇到的问题和错误[6,7].在软件维护期间,我们可以通过复杂性度量和比较,选择成本有效性的修改手段.软件修改和影响是软件维护和演化过程中不可缺少的一部分,因此,整个回归测试的复杂性可以分为修改复杂性、影响复杂性以及重测复杂性.软件修改可以在不同层面上,我们分别从构件和系统两个层面讨论回归测试的复杂

性因素.

图 1 利用鱼骨头图(fishbone graph)[8]描述了构件层面的回归测试复杂性因素.从图中看到,构件层面主要存在 4 个修改复杂性因素:内部数据;内部功能;API 功能;端口(port).

其中,API 功能表示外界对构件的调用接口,端口表示构件本身对其他构件的调用或者信息传递.这 4 个因素在构件软件维护中经常发生修改.在构件修改中,内部数据和函数是经常变动的.API 功能的修改和升级是构件新版本发布的重要变化.构件的调用关系也经常通过端口而发生改变.软件修改方式主要分为添加(add)、删除(delete)和改动(change),比如说可以在新版本的构件中添加、删除或者改动 API 的功能.

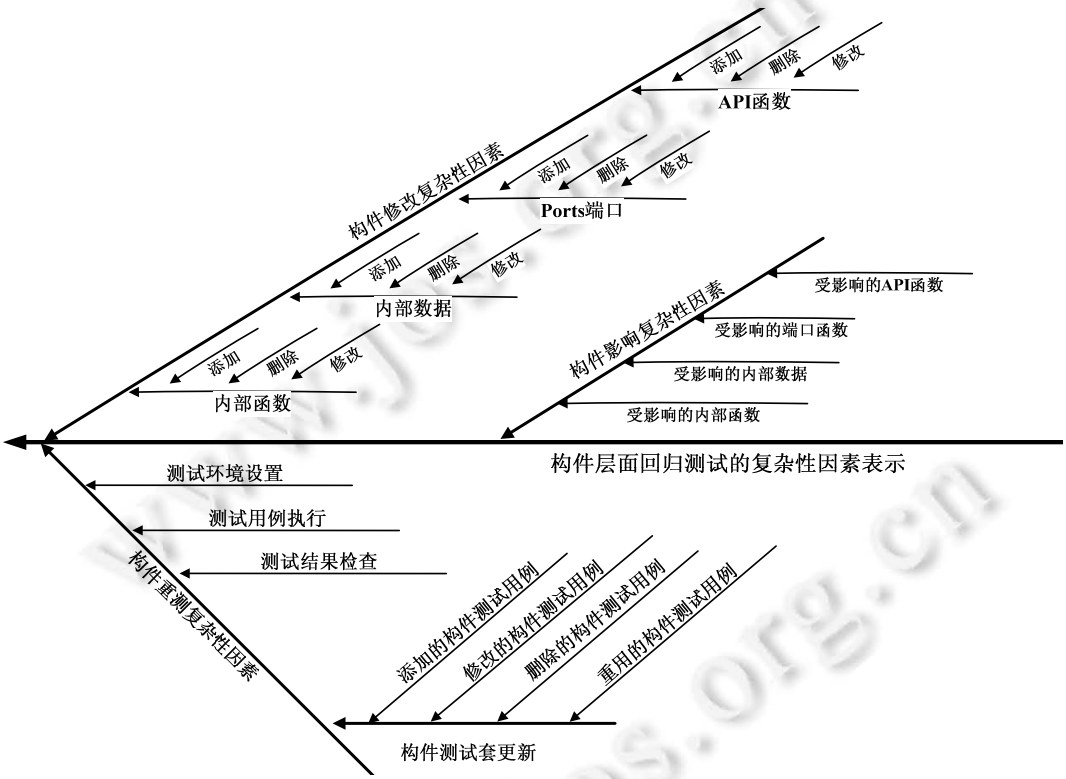



Fig.1 Regression testing complexity factors at component level

图 1 构件层面的回归测试复杂性因素

同理,对于构件受修改的影响,也主要发生在这几个因素之中,因此,我们把受影响的内部函数、内部功能、API 功能和端口作为影响分析的复杂性因素.

下面我们分析构件层面重测的复杂性因素.从测试角度看,重测会涉及到很多复杂性因素,比如测试环境的搭建、测试用例的执行、测试结果的检查等等.如果不考虑一般测试的复杂性,只从回归测试的角度考虑,重测复杂性主要与测试套的更新相关,因为测试套的更新反映了测试用例的变化情况,包括测试用例的添加、删除、修改和重用.不同的测试用例更新方式会造成不同的复杂性.通常来说,添加测试用例的复杂性要比重用测试用例的复杂性要高.

图 2 是系统层面的回归测试复杂性因素.当系统中添加一个构件的时候,就会对系统的组合关系产生影响.构件的功能或者数据的修改也可能会对构件之间的消息传递产生影响.另外,当系统中的可配置功能、数据或者环境发生变化时,配置关系也会发生改变.因此,我们总结的系统层面的主要修改因素有组合(composition)、配置(configuration)以及消息(message).同样,我们将受修改影响的组合、配置和消息作为影响分析的复杂性因素.

与构件层面类似,系统层面的重测复杂性因素和系统测试套的更新相关,我们总结为添加的系统测试用例、删除的系统测试用例、修改的系统测试用例以及重用的系统测试用例这 4 个因素.

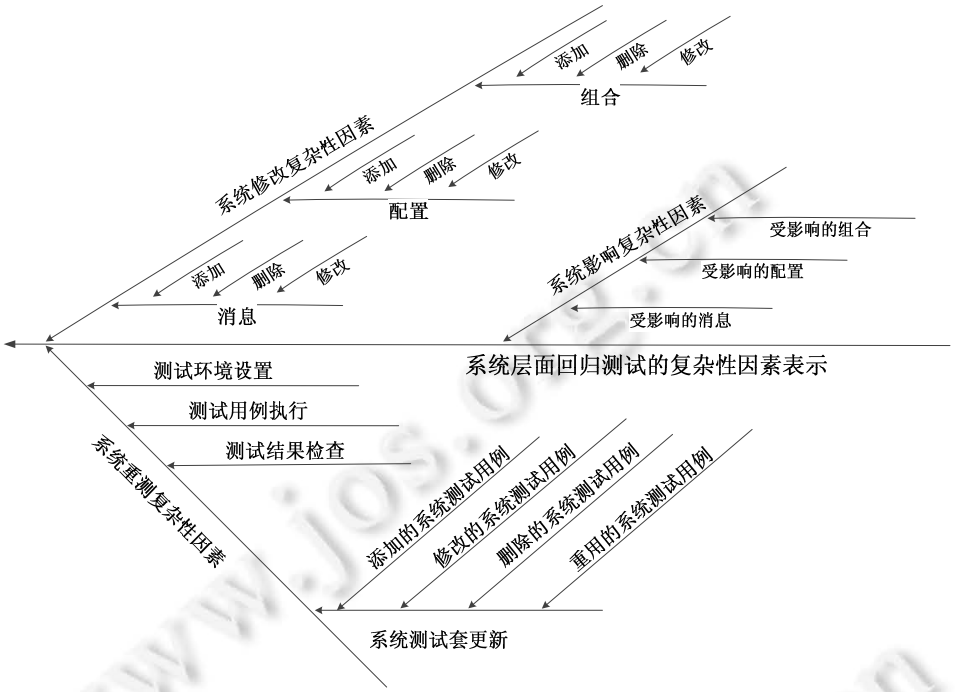



Fig.2 Regression testing complexity factors at system level

图 2 系统层面的回归测试复杂性因素

## 2 构件软件的回归测试复杂性度量框架

使用复杂性度量可以比较不同回归测试过程的复杂性,从而选择具有成本有效性的修改、影响以及重测手段.当前的相关工作中还没有从软件维护的角度来研究回归测试的复杂性.在本节中,我们将提出一种回归测试复杂性的度量框架,该度量框架包括基于图的模型和形式化度量计算两个部分.

### 2.1 基于图的模型表示

回归测试复杂性主要依赖于不同的复杂性因素.如何将这些因素应用到复杂性的分析和计算中,是本文研究的主题.我们从以下的研究动机出发,提出一种度量框架:

(1) 如何有效表示多复杂性因素?

(2) 如何动态表示复杂性计算结果?

(3) 如何可视化地体现各个复杂性因素对于构件或者系统的复杂性影响?

受 McCall 质量保证模型的启发,我们提出一种回归测试复杂性模型——复杂性图模型(complexity graphic model,简称 CGM).该模型由雷达图来表示,其中,每一个方向表示复杂性因素,从图的原点到极点之间的距离表示每个所涉及到的复杂性因素复杂性的计算值.极点之间的连线构成一个多边形,而多边形的面积可以用来表示最后总的复杂性计算结果的值.

根据文献[9]中提出的多边形面积计算方法,我们可以利用下面的公式来表达回归测试复杂性(regression testing complexity,简称 RTC):

$$RTC = \frac{1}{2}\sin\frac{2\pi}{n}\sum_{i=1}^{n} a_i \times a_{i+1} \tag{1}$$

其中,$a_i$表示不同的复杂性因素,$n$表示所涉及到的复杂性因素数目.每个复杂性因素对应于CGM中的一条链接.

基于图的模型可以表示不同的复杂性因素以及它们的复杂性计算值.与单一的线性复杂性计算方式相比,本方法可以有效体现多复杂性因素以及复杂性之间的动态监控和分析比较,还能够支持复杂性分析自动化工具的开发.图 3 是一些回归测试复杂性模型示例.

图 3(a)~图 3(d)分别表示回归测试中的修改、影响、重测和测试套更新的复杂性计算模型.注意:图 3(c)表示通常情况下的重测复杂性因素;而图 3(d)是我们研究的回归测试重测复杂性模型,也就是测试套更新的复杂性计算模型.

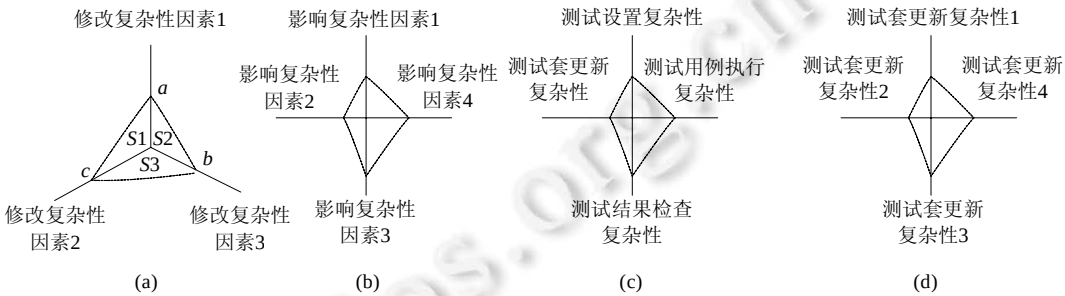



Fig.3 Samples of regression testing complexity model

图 3 构件软件回归测试复杂性度量模型示例

举例来说,在图 3(a)中,假设本次修改涉及到 3 个复杂性因素 $a,b,c$,那么根据公式(1),修改的复杂性可以计算如下:

$$Area = S1 + S2 + S3 = \frac{1}{2}\sin 120(ac + ab + bc).$$

同样,我们可以计算图 3(b)中的影响复杂性和图 3(d)中的测试套更新复杂性.

### 2.2 构件层面回归测试复杂性度量

度量构件层面的回归测试复杂性有如下两个步骤:

(1) 度量每个因素(修改、影响或者测试更新)的复杂性;

(2) 根据图形模型度量总的多因素复杂性.

#### 2.2.1 构件修改复杂性

首先,我们考虑如何利用上节所识别和分析的复杂性因素来有效计算修改的复杂性.直观上来看,修改复杂性和构件或者系统中发生修改的数目应该成正比.另外,不同的修改因素可能导致不同的修改复杂性.比如,构件 API 的修改通常比内部功能函数的修改带来的复杂性要高.因此,在度量中要考虑赋予它们更高的复杂性权重.对于每个复杂性因素来说,我们还需要考虑一个修改的百分比问题,也就是修改的数目占该修改因素所对应数目的百分比,从而平衡整个程序的修改复杂性.比如说:

- 给定 4 个发生修改的 API,构件中总的 API 数目是 10,那么修改的百分比就是 4/10=40%;
- 如果删除 2 个 API,则百分比就是 2/10=20%;
- 如果添加 5 个 API,那么百分比是 5/(5+10)=33.3%.

因此,我们在分析和计算修改复杂性时,需要考虑修改的数目、所占的百分比以及不同修改因素的权重.

目前有不少关于成本或者复杂性的估算模型,比较经典的软件开发成本估算模型有功能点(function point,简称 FP)模型[10].它首先列举出影响软件工程开发成本的一些主要因素,然后给每个因素赋予一定的权重,通过相乘的方式计算软件功能点.后来,功能点模型也在软件维护中得到了广泛研究和应用[11].受该模型的启迪,我们采用功能点的类似表达方式,通过将修改数目、修改百分比以及权重相乘的初步计算方式得到构件修改复杂性的计算值.根据软件开发和维护经验,添加代码通常比修改代码要复杂,修改代码可能比删除代码要复杂.另

外,不同的修改复杂性因素会导致不同的修改复杂性,因此,我们赋予不同的权重.通过大量的实验研究,给出了不同修改复杂性因素权重的经验值.

表 1 是构件软件回归测试复杂性因素的度量计算方法,其中的权值是根据实验中经验数据所得到.这些值是可以调整的,也就是软件维护人员在实践过程中可以根据需要和经验反馈调整权重.

**Table 1** Complexity metrics for various change factors

**表 1** 不同修改因素的复杂性度量

| 修改因素 | 修改类型 | 修改数量 | 权重 | 修改比值 | 修改复杂性 |
|---|---|---|---|---|---|
| 内部数据 | 添加<br>删除<br>修改 | × | 4<br>1<br>2 | × | = |
| 内部函数 | 添加<br>删除<br>修改 | × | 4<br>1<br>2 | × | = |
| API 函数 | 添加<br>删除<br>修改 | × | 6<br>2<br>4 | × | = |
| 端口 | 添加<br>删除<br>修改 | × | 5<br>2<br>3 | × | = |
| 消息 | 添加<br>删除<br>修改 | × | 5<br>2<br>3 | × | = |
| 组合 | 添加<br>删除<br>修改 | × | 6<br>3<br>4 | × | = |
| 配置 | 添加<br>删除<br>修改 | × | 6<br>3<br>4 | × | = |

对于每个修改复杂性因素 $C_i$,其在回归测试中的修改复杂性可以通过公式(2)进行计算:

$$\begin{aligned} Complexcomp(C_i) = &|F^a_{comp}(C_i)|\times W^a_{comp}(C_i)\times P^a_{comp}(C_i) + |F^d_{comp}(C_i)|\times W^d_{comp}(C_i)\times P^d_{comp}(C_i) \\ &+ |F^c_{comp}(C_i)|\times W^c_{comp}(C_i)\times P^c_{comp}(C_i) \end{aligned} \tag{2}$$

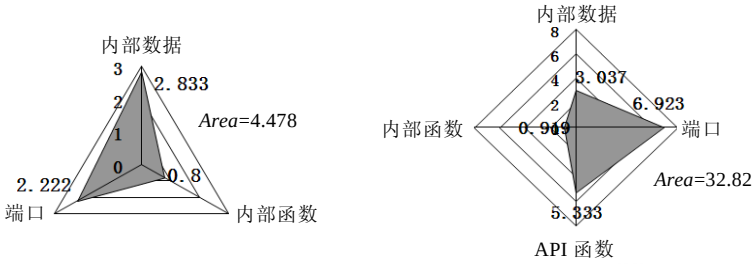
其中,$a,d,c$ 分别表示添加、删除和修改,$|F^a_{comp}(C_i)|$ 表示修改因素 $C_i$ 所添加的数目,$F,W,P$ 的右下标 $comp$ 表示构件层面的修改;$W^a_{comp}(C_i), W^d_{comp}(C_i)$ 和 $W^c_{comp}(C_i)$ 表示相应的权值;$P^a_{comp}(C_i), P^d_{comp}(C_i)$ 和 $P^c_{comp}(C_i)$ 表示修改因素 $C_i$ 被修改的数目占总体的百分比

上面是对每个修改因素的复杂性进行计算,我们知道,每个被修改的构件中可能存在多种修改因素.根据前面的回归测试复杂性图模型 CGM,可以对具有多复杂性修改因素的构件进行修改复杂性分析和计算.这里,我们针对构件 comp 提出一种回归测试修改复杂性度量 $RTCC(comp)$.假设构件 comp 中存在 $n(n\geqslant 3)$个修改因素 $C_i(1\leqslant i\leqslant n)$,$RTCC(comp)$可以进行如下计算:

$$RTCC(comp) = \frac{1}{2}\sin\frac{2\pi}{n}\sum_{i=1}^{n} Complex_{comp}(c_i)\times Complex_{comp}(c_{i+1}) \tag{3}$$

其中,$Complex_{comp}(c_i)$表示任意修改因素 $c_i$ 的复杂性,我们这里规定 $c_{n+1}=c_1$.

图 4(a)和图 4(b)是两个构件的修改复杂性的雷达图模型示例.在电梯系统中,我们修改了构件 Meta-Controller 和 FloorPanel.例如在构件 Meta-Controller 中有 3 种修改因素:内部数据$(c_1)$;内部函数$(c_2)$;端口$(c_3)$.




(a) 元控制构件 (b) 楼层控制板构件

Fig.4 Samples of change complexity models of meta-controller and floor panel

图 4 构件修改复杂性的雷达图模型示例

表 2 显示了每个修改因素的复杂性计算情况.构件总的修改复杂性由三角形中的阴影部分面积表示.下面举例说明图 4(a)中具体的修改复杂性计算过程,主要分为两步:

- 第 1 步:每个修改因素的复杂性计算,具体计算过程如下:

$$Complex_{comp}(c_1)=(2\times4\times33.33\%)+(1\times1\times16.67\%)=2.834,$$

$$Complex_{comp}(c_2)=2\times2\times20\%=0.800,$$

$$Complex_{comp}(c_3)=2\times5\times22.22\%=2.222;$$

- 第 2 步:总的修改复杂性计算:

$$\begin{aligned}RTCC(metacontroller)&=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{i=1}^{n}Complex_{comp}(c_i)\times Complex_{comp}(c_{i+1})\\&=\frac{1}{2}\times(2.833\times0.8+0.8\times2.222+2.222\times2.833)\sin\left(\frac{360}{3}\right)\\&=4.478.\end{aligned}$$

在图 4(b)中,构件 FloorPanel 的修改存在 4 种修改复杂性因素:内部数据、内部函数、端口和 API.因此,根据 CGM 模型,其修改复杂性可由一个四边形中的阴影部分面积来表示.具体的计算过程和上面类似.

**Table 2** A sample complexity metrics for change factors

**表 2** 修改因素的复杂性计算示例

| 修改因素 | 修改类型 | 修改数量 | 权重 | 修改比值(%) | 修改复杂性 |
|---|---|---|---|---|---|
| $Complex_{comp}(c_1)$ | 添加 | 2 | 4 | 33.33 | 2.667 |
| | 删除 | 1 | 1 | 16.67 | 0.167 |
| $Complex_{comp}(c_2)$ | 修改 | 2 | 2 | 20 | 0.800 |
| $Complex_{comp}(c_3)$ | 添加 | 2 | 5 | 22.2 | 2.222 |

2.2.2 构件影响复杂性

当构件发生修改后,其内部数据、函数、API、端口都有可能受到影响.因此,这些因素也是主要的修改影响复杂性因素.对于不同的影响复杂性因素,我们仍然采用 3 个主要的复杂性计算参数:每个影响因素的数目;复杂性权重;受影响的百分比.表 3 给出了不同影响因素的复杂性度量具体计算方式.

对于每个影响复杂性因素 $I_j$,其在回归测试中的影响复杂性可以通过如下公式进行计算:

$$Complex_{comp}(I_j)=|F_{comp}(I_j)|\times W_{comp}(I_j)\times P_{comp}(I_j) \quad (4)$$

其中,$|F_{comp}(I_j)|$表示影响因素 $I_j$的数目,$W_{comp}(I_j)$表示相应的权值,$P_{comp}(I_j)$表示影响因素 $I_j$的数目占总体的百分比.

这里,我们针对构件 comp 提出一种回归测试影响复杂性度量 $RTIC(comp)$.假设构件 comp 中存在 $n(n\geqslant3)$个影响因素 $I_j(1\leqslant j\leqslant n)$,$RTIC(comp)$可以进行如下计算:

$$RTIC(comp)=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{j=1}^{n}Complex_{comp}(I_j)\times Complex_{comp}(I_{j+1}) \quad (5)$$

其中,$Complex_{comp}(I_j)$表示任意影响因素 $I_i$ 的复杂性,这里规定 $I_{n+1}=I_1$.

**Table 3** Complexity metrics for different impact factors

**表 3** 影响因素的复杂性计算

| 影响因素 | 影响数量 | 权重 | 影响比值 | 影响复杂性 |
|---|---|---|---|---|
| 受影响的内部数据 | × | 1 | × | = |
| 受影响的内部函数 | × | 2 | × | = |
| 受影响的 API 函数 | × | 4 | × | = |
| 受影响的端口 | × | 3 | × | = |
| 受影响的消息 | × | 4 | × | = |
| 受影响的组合 | × | 5 | × | = |
| 受影响的配置 | × | 5 | × | = |

下面举例说明每个影响因素复杂性的计算过程.图 5(a)是构件 Meta-Controller 的影响复杂性的雷达图模型示例.它的影响复杂性主要依赖于 3 个因素:受影响的内部数据($I_1$)、内部函数($I_2$)、端口 API($I_3$).


(a) 元控制构件 (b) 构件系统

Fig.5 Samples of impact complexity models of meta-controller and system

图 5 构件及系统影响复杂性的雷达图模型示例

那么,其影响复杂性可以计算如下:

$$Complex_{comp}(I_1)=|F_{comp}(I_1)|\times W_{comp}(I_1)\times P_{comp}(I_1)=1\times 1\times 16.67\%=0.167,$$

$$Complex_{comp}(I_2)=|F_{comp}(I_2)|\times W_{comp}(I_2)\times P_{comp}(I_2)=1\times 2\times 10\%=0.200,$$

$$Complex_{comp}(I_3)=|F_{comp}(3)|\times W_{comp}(I_3)\times P_{comp}(I_3)=1\times 4\times 11.11\%=0.444;$$

$$RTIC(metacontroller)=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{j=1}^{n}Complex_{comp}(I_i)\times Complex_{comp}(c_{i+1})$$

$$=\frac{1}{2}\times(0.167\times 0.2+0.2\times 0.444+0.444\times 0.167)\times\sin\left(\frac{360}{3}\right)$$

$$=0.084.$$

2.2.3 构件测试套更新复杂性

重测的复杂性主要依赖于测试套中添加、删除、修改、重用的测试用例.在文献[12]中提到,测试复杂性主要是由能够证明程序正确性的测试数据数目来度量.因此,我们也将测试用例的数目作为主要的测试复杂性因素.在文献[13]中,为了简化测试成本分析模型,研究者假定所有的测试具有同样的成本.在本方法中,我们也基于两个假设:测试套中的测试用例是相互独立的;每个测试用例具有相对平均的复杂性.根据前面分析,我们可以总结出测试套更新的复杂性计算参数有:测试用例的数目(添加、删除、修改或者重用)、测试用例的权重、测试用例的百分比.

表 4 的上半部分是对测试套更新的每个具体因素,也就是 4 种构件测试用例更新方式的复杂性计算.表中的权值是根据实验得到的经验值.

**Table 4** Complexity metrics for diverse factors of test suite refreshment

**表 4** 不同测试套更新因素的复杂性计算

| 测试套更新因素 | 测试用例数量 | 权重 | 测试用例比值 | 重测复杂性 |
|---|---|---|---|---|
| 添加的构件测试用例 | × | 4 | × | = |
| 修改的构件测试用例 | × | 3 | × | = |
| 删除的构件测试用例 | × | 1 | × | = |
| 重用的构件测试用例 | × | 2 | × | = |
| 添加的系统测试用例 | × | 6 | × | = |
| 修改的系统测试用例 | × | 5 | × | = |
| 删除的系统测试用例 | × | 2 | × | = |
| 重用的系统测试用例 | × | 3 | × | = |

对于每个测试更新复杂性因素,在测试用例添加、删除、修改、重用的复杂性分别表示为 $Complex^{a}_{comp}(T_k)$, $Complex^{d}_{comp}(T_k), Complex^{c}_{comp}(T_k), Complex^{r}_{comp}(T_k)$,可以通过如下公式进行计算:

$$\left.\begin{aligned} Complex^{a}_{comp}(T_k) &= |F^{a}_{comp}(T_k)| \times W^{a}_{comp}(T_k) \times P^{a}_{comp}(T_k) \\ Complex^{d}_{comp}(T_k) &= |F^{d}_{comp}(T_k)| \times W^{d}_{comp}(T_k) \times P^{d}_{comp}(T_k) \\ Complex^{c}_{comp}(T_k) &= |F^{c}_{comp}(T_k)| \times W^{c}_{comp}(T_k) \times P^{c}_{comp}(T_k) \\ Complex^{r}_{comp}(T_k) &= |F^{r}_{comp}(T_k)| \times W^{r}_{comp}(T_k) \times P^{r}_{comp}(T_k) \end{aligned}\right\} \tag{6}$$

如果回归测试中涉及到 3 个以上的复杂性因素,我们需要基于雷达图模型 CGM 来进行建模和计算.类似于构件修改和影响复杂性的分析,我们提出下面的多因素测试套更新的复杂性计算公式:

$$RTTC(comp) = \frac{1}{2}\sin\frac{2\pi}{n}\sum_{k=1}^{n=3,4} Complex_{comp}(T_k) \times Complex_{comp}(T_{k+1}) \tag{7}$$

图 6 是构件 FloorPanel 的重测复杂性的雷达图模型示例,其中涉及到 3 种测试用例更新方式,也就是 3 个复杂性因素:添加测试用例($T_1$)、删除测试用例($T_2$)、修改测试用例($T_3$).具体的计算过程如下:

$$Complex_{comp}(T_1) = |F^{a}_{comp}(T_1)| \times W^{a}_{comp}(T_1) \times P^{a}_{comp}(T_1) = 7 \times 4 \times 50\% = 14,$$

$$Complex_{comp}(T_2) = |F^{d}_{comp}(T_2)| \times W^{d}_{comp}(T_2) \times P^{d}_{comp}(T_2) = 2 \times 1 \times 16.7\% = 0.33,$$

$$Complex_{comp}(T_3) = |F^{c}_{comp}(T_3)| \times W^{c}_{comp}(T_3) \times P^{c}_{comp}(T_3) = 7 \times 3 \times 50\% = 10.5.$$

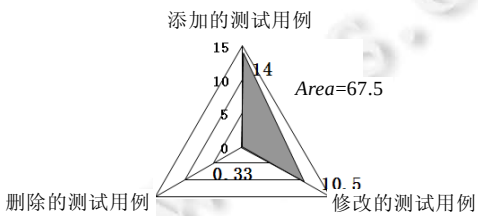



Fig.6 A sample model of component test suite refreshment complexity for Floorpanel retest

图 6 构件测试套更新的复杂性计算雷达图模型

于是,总的测试套更新复杂性是:

$$\begin{aligned} RTTC(FloorPanel) &= \frac{1}{2}\sin\frac{2\pi}{n}\sum_{k=1}^{n=3} Complex_{comp}(T_k) \times Complex_{comp}(T_{k+1}) \\ &= \frac{1}{2} \times \sin\frac{360}{3} \times (14 \times 0.33 + 0.33 \times 10.5 + 10.5 \times 14) \\ &= 67.50. \end{aligned}$$

### 2.3 系统层面回归测试复杂性度量

在第 2.2 节,我们讨论了构件层面回归测试的复杂性度量.在系统层面,采用类似的度量框架来进行分析和

计算.

我们分析了系统层面的修改主要发生在组合关系、消息传递以及架构配置的修改.所以,这 3 个因素可以看成是系统修改的复杂性因素.对于每个系统修改因素 $C_i$,其复杂性可以使用公式(8)计算:

$$Complexs(C_i)=|F_s^a(C_i)|\times W_s^a(C_i)\times P_s^a(C_i)+|F_s^d(C_i)|\times W_s^d(C_i)\times P_s^d(C_i)+|F_s^c(C_i)|\times W_s^c(C_i)\times P_s^c(C_i) \tag{8}$$

公式中的符号与构件层面是类似的,因此这里就不再具体解释.

对于多因素(3 个以上)的系统修改复杂性,基于雷达图模型给出计算方式.

这里,针对系统提出一种回归测试修改复杂性度量 $RTCC(system)$.假设系统中存在 $n(n\geqslant 3)$个修改因素 $C_i(1\leqslant i\leqslant n)$,$RTCC(system)$可以用下面的公式进行计算:

$$RTCC(system)=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{i=1}^{n}Complex_s(c_i)\times Complex_s(c_{i+1}) \tag{9}$$

构件或者系统的修改都有可能对系统产生影响,因此,需要分析系统层面的影响复杂性.前面分析的系统影响复杂性是受修改影响的组合关系、消息以及配置.对于每个系统影响复杂性因素 $I_j$,其在回归测试中的影响复杂性可以通过如下公式进行计算:

$$Complex_s(I_j)=|F_s(I_j)|\times W_s(I_j)\times P_s(I_j) \tag{10}$$

其中,$|F_s(I_j)|$表示影响因素 $I_j$ 的数目,$W_s(I_j)$表示相应的权值,$P_s(I_j)$表示影响因素 $I_j$ 的数目占总体的百分比.

这里,我们针对系统提出一种回归测试系统影响复杂性度量 $RTIC(system)$.假设系统中存在 $n(n\geqslant 3)$个影响因素 $I_j(1\leqslant j\leqslant n)$,$RTIC(system)$可以进行如下计算:

$$RTIC(system)=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{j=1}^{n}Complex_s(I_j)\times Complex_s(I_{j+1}) \tag{11}$$

其中,$Complex_s(I_j)$表示任意修改因素 $c_i$ 的复杂性,这里规定 $I_{n+1}=I_1$.

图 5(b)表示系统影响复杂性的雷达图模型,其中有 3 个影响复杂性因素.与构件测试套类似,系统层面的测试套更新复杂性也采用这种方式.

第 2.2 节中,表 4 的下半部分是对系统测试套更新的每个具体因素,也就是 4 种系统测试用例更新方式的复杂性计算.权值是根据我们的实验经验所得到.对于每个系统测试套更新复杂性因素,在测试用例添加、删除、修改、重用的复杂性分别表示为 $Complex_s^a(T_k)$,$Complex_s^d(T_k)$,$Complex_s^c(T_k)$,$Complex_s^r(T_k)$,可以通过如下公式进行计算:

$$\left.\begin{aligned}Complex_s^a(T_k)&=|F_s^a(T_k)|\times W_s^a(T_k)\times P_s^a(T_k)\\Complex_s^d(T_k)&=|F_s^d(T_k)|\times W_s^d(T_k)\times P_s^d(T_k)\\Complex_s^c(T_k)&=|F_s^c(T_k)|\times W_s^c(T_k)\times P_s^c(T_k)\\Complex_s^r(T_k)&=|F_s^r(T_k)|\times W_s^r(T_k)\times P_s^r(T_k)\end{aligned}\right\} \tag{12}$$

如果回归测试中涉及到 3 个以上的系统测试更新复杂性因素,我们需要基于雷达图模型 CGM 来进行建模和计算.类似于构件修改和影响复杂性分析,提出下面的测试更新多因素复杂性计算公式:

$$RTTC(system)=\frac{1}{2}\sin\frac{2\pi}{n}\sum_{k=1}^{n=3,4}Complex_s(T_k)\times Complex_s(T_{k+1}) \tag{13}$$

## 3 实验研究

在实验分析中,我们将基于雷达图的回归测试复杂性度量框架应用到一个基于构件的系统中.该系统是一种电梯模拟系统,由美国加州 SJSU 大学计算机工程学院学生开发.我们在前期工作中也使用了该系统的某些版本作为实验对象[4,5].

图 7 是该系统的构件结构图.该系统是由管理控制、车载、车载控制、电梯门、用户面板、楼层面板等构件组合而成.

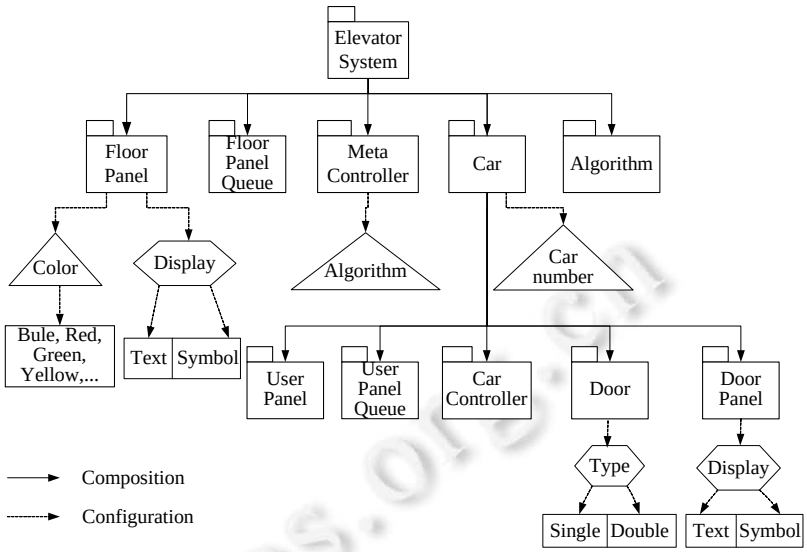



Fig.7 A component diagram for the component-based elevator simulation system

图 7 电梯模拟系统的构件结构图

### 3.1 实验描述和实验控制

我们在新版本的电梯系统中实施了如下的修改需求:

a) 在构件中添加楼层显示器(floor indicator),使得控制面板和车载内用户面板,可以显示电梯当前到达的楼层;

b) 添加电梯可以只停奇数和偶数层的功能.

我们使用前期工作中的回归测试方法对修改后的电梯系统进行系统化的回归测试[4,5],包括修改识别、影响分析、测试用例更新等.然后,利用本文介绍的模型和度量方式进行复杂性分析和度量.实验数据主要来源于两个部分:

(1) 更新的设计文档.用来记录构件和系统的修改、影响情况以及度量所得到的复杂性结果;

(2) 重测文档.用来记录添加、删除、修改和重用的测试用例以及度量所得到的测试套更新复杂性.

为了减少其他复杂性因素的影响,我们进行了一些实验控制.在所有实验组中,我们选取了 3 组结果作为实验分析对象.实验组成员由软件工程一年级下学期的研究生所组成,每组 3~4 个成员,具有类似的知识和能力背景,本科都毕业于软件工程和计算机系.实验中,修改构件的人员具有相同的程序经验,测试经验则主要来自于同一门构件软件测试课程,用来减少实验复杂性结果的人为差异.根据统一的修改需求,每组都实施了自己独立的具体修改方式和手段.组员使用了基于前期工作的修改影响分析方法[4],得到构件和系统层面的影响分析结果.首先,对实验对象修改前后的版本进行建模,包括修改前后版本的功能依赖图、数据依赖图模型、API 模型、交互图模型和组合配置模型;然后,在构件和系统模型上识别出相应的修改点;接着,利用修改影响分析方法对修改点进行影响分析和计算,找到受修改影响的构件内部功能和数据、API、交互关系、架构关系等.在修改之前的版本,我们提供了达到一定测试充分性的测试套,其中包括构件测试用例和系统测试用例.在修改之后,基于前面的修改影响分析,实验组员更新了测试用例,得到新版本的测试用例集合.对于测试复杂性的其他因素(图 3(c)中的表示),我们在实验前后基本保持统一不变,减少其他复杂性因素的干扰.

### 3.2 实验结果分析

利用本文提出的度量框架,我们得到了对上述回归测试进行复杂性度量的结果.表 5 表示每个构件的修改复杂性值.

**Table 5** Results of change complexity metric

**表 5** 修改复杂性度量结果

| Group No. | UserPanel | FloorPanel | Algorithm | Car | Car controller | Admin console | Metacontrloller | System |
|---|---|---|---|---|---|---|---|---|
| G1 | 0.78 | 5.82 | – | 3.10 | 0.46 | 6.23 | 4.48 | 10.00 |
| G2 | 2.24 | 3.10 | 4.67 | 2.59 | 3.50 | 1.18 | – | 6.99 |
| G3 | 1.36 | 67.03 | 1.98 | 1.86 | – | 3.11 | 1.67 | 8.93 |

每个实验小组对于不同的构件都有不同的修改复杂性:第 1 组在构件 Car,AdminConsole 和 Metacontroller 中的修改复杂性最高,计算值分别为 3.10,6.23 和 4.48;第 2 组在构件 UserPanel 和 Algorithm 中的修改复杂性最高,分别是 2.24 和 4.67.在构件 FloorPanel 中,第 3 组的修改复杂性比其他两组都高出很多,复杂性值为 67.03,第 1 组是 5.82,第 2 组是 3.10.原因是第 3 组在构件 FloorPanel 中添加了一个新构件 FloorIndicator.在系统层面上,第 3 组的修改复杂性是最高的,值为 10.

表 6 是影响复杂性度量的实验结果.从中可以看出:

- 在构件层面,第 1 组在很多构件中的影响复杂性都相对比较低,比如在构件 UserPanel 中的影响复杂性是 0.154,在 AdminConsole 中的影响复杂性是 0.082,在 Metacontroller 中的影响复杂性是 0.084;第 2 组则在构件 FloorPanel,CarController,AdminConsole,Metacontroller 中具有最高的影响复杂性;
- 在系统层面,第 1 组有最高的系统影响复杂性值 12,而第 2 组最低,其值为 1.5.

**Table 6** Results of impact complexity metrics

**表 6** 影响复杂性度量结果

| Group No. | UserPanel | FloorPanel | Algorithm | Car | Car controller | Admin console | Metacontrloller | UserPanel queue | System |
|---|---|---|---|---|---|---|---|---|---|
| G1 | 0.154 | – | – | 0.256 | 0.091 | 0.082 | 0.084 | 0.941 | 12.00 |
| G2 | 0.712 | 0.286 | – | 0.042 | 1 | 1.653 | 1.333 | – | 1.50 |
| G3 | 1.82 | 0.088 | 8.66 | 1.133 | – | 0.27 | 0.653 | – | 9.75 |

测试套更新的复杂性度量结果见表 7.我们在后面将和修改、影响的复杂性结合讨论.

**Table 7** Results of test suite refreshment complexity metrics

**表 7** 测试套更新复杂性的度量结果

| Group No. | UserPanel | FloorPanel | Algorithm | System |
|---|---|---|---|---|
| G1 | 21.14 | 82.30 | – | 146.60 |
| G2 | 15.03 | 19.05 | 53.10 | 122.80 |
| G3 | 13.20 | 136.20 | 29.10 | 71.50 |

### 3.3 实验验证

上述实验表明了复杂性度量框架在构件软件回归测试中应用的可行性,下面将验证该方法的有效性.我们知道,复杂性是软件维护和测试成本的一个主要因素.在我们的实验中,将通过复杂性和实际的维护及测试成本作比较来进行验证,并记录整个回归测试所花费的成本.实际成本包括修改、影响以及测试套更新的成本.3 个实验组分别报告了构件和系统两个层面的实际成本.我们以人-小时(person-hour)为基本单位来度量成本.其中,

- 修改成本是以修改识别所花费的时间来度量;
- 修改影响的成本以实际的修改所花费的时间来度量;
- 测试成本则根据测试套的更新,包括测试执行时间、测试覆盖分析和测试结果检查.

表 8 是实验验证结果,表中记录了各个实验组在不同构件以及系统中的复杂性及成本.

从表 8 中可以看出:在大部分情况下,回归测试复杂性越高,所花费的成本越高.比如:第 3 组在构件 FloorPanel 中有最高的修改复杂性,同时也具有最大的成本;而第 1 组在构件 UserPanel 中的修改复杂性最低,而且修改花费成本也最小.表中的一些测试成本和复杂性看起来并不是成比例的,例如,第 3 组的系统测试复杂性高于第 1 组,但这两组报告了类似的测试成本:35 和 37.这是由于外部测试环境复杂性的存在,在实际应用中,我

们的重测复杂性的精度也受到影响.因此在未来工作中,我们还需要对我们提出的复杂性度量框架进行改善,从而提高复杂性度量精度.

**Table 8** Comparison between complexity metrics and actual effort record

**表 8** 复杂性度量结果和实际花费成本比较

| | CC | CE | IC | IE | TC | TE |
|---|---|---|---|---|---|---|
| G1-UserPanel | 0.78 | 7 | 0.15 | 3 | 21.14 | 17 |
| G1-FloorPanel | 5.82 | 16 | – | – | 82.30 | 20 |
| G1-Algorithm | – | – | – | – | – | – |
| G1-System | 10 | 18 | 12 | 20 | 46.60 | 35 |
| G2-UserPanel | 2.24 | 22 | 0.17 | 2 | 15.03 | 12 |
| G2-FloorPanel | 3.10 | 25 | 0.29 | 3 | 19.05 | 15 |
| G2-Algorithm | 4.67 | 28 | – | – | 53.10 | 32 |
| G2-System | 6.99 | 37 | 1.50 | 6 | 122.80 | 46 |
| G3-UserPanel | 1.36 | 15 | 1.82 | 21 | 13.20 | 11 |
| G3-FloorPanel | 67.03 | 52 | 0.09 | 1 | 136.20 | 45 |
| G3-Algorithm | 1.98 | 17 | 8.66 | 21 | 29.10 | 31 |
| G3-System | 8.93 | 49 | 9.75 | 42 | 71.50 | 37 |

注:CC:修改复杂性;CE:修改成本;IC:影响复杂性;IE:影响成本;TC:测试套更新复杂性;TE:测试套更新成本

通过回归测试的成本比较,初步表明所提出的度量框架可以有效支持复杂性分析和度量.本实验验证还有更多的工作可以继续,比如,要进一步研究回归测试复杂性成本和复杂性之间的关联,需要更多的统计分析,像回归分析等.另外,我们可以根据大量统计结果分析,开发带成本预测功能的原型工具.

### 3.4 实验结果观察和启发

针对同样的修改需求,3 个实验组作出了不同的具体修改实施,带来了不同的影响和测试套更新结果.这也导致了差异比较明显的复杂性度量结果.经过实验分析和验证,我们观察并得到了一些比较有启发性的结果.

**观察结论 1**. 对于一个被修改的构件,它的影响复杂性和修改复杂性不一定成正比.

如图 8 所示,构件 UserPanelQueue 在第 1 组中没有被修改,但是在图 9 中可以看出,该构件有很高的影响复杂性,占据了总影响复杂性的 58%.这是因为其中大部分的影响是由其他具有交互关系的构件修改所导致,比如构件 CarController 的修改.在第 2 组中,构件 Algorithm 的修改复杂性值是 4.67,占据总修改复杂性的 27%.但值得注意的是,第 2 组在构件 Algorithm 中没有报告任何影响复杂性.我们分析,导致这个现象的主要原因是:这些修改大部分发生在构件内部,并且这些修改没有影响很多构件 API 和交互关系.

**观察结论 2**. 对于一个被修改的构件,它的测试套更新复杂性不一定与修改以及影响复杂性成正比.

第 2 组和第 3 组在构件 Algorithm 中的修改复杂性值分别是 4.67(占总修改复杂性的 27%)和 1.98(占总修改复杂性的 3%),如图 8 所示.但在图 9 中,第 3 组报告的影响复杂性值是 8.66(占总影响复杂性的 69%),但第 2 组没有报告影响复杂性.因此,第 3 组的修改和影响复杂性之和要大于第 2 组.但在表 7 中我们发现:第 2 组在构件 Algorithm 报告的测试套更新复杂性值是 53.1,大于第 3 组的复杂性值 29.1.原因是,实验组采取了不同的测试方法生成新的测试用例.比如说,第 3 组采用的是基于路径的测试方法,而第 2 组使用了分支覆盖测试方法.通过对构件 Algorithm 的内部代码检查,我们也发现第 2 组在其内部的修改涉及到了比较多的分支.这个实验观察结论启示我们,不同的测试方法也可能导致不同的重测复杂性.

上面的实验观察主要从单个构件的角度来看,下面通过所有涉及到的构件修改、影响以及测试套更新复杂性来进行系统性的观察.图 10 和图 11 给出了构件和系统层面的修改和影响复杂性总结.如图 10 所示:第 3 组在构件层面报告了最高的修改复杂性值 77.01,这是由于其添加了一个新的构件 FloorIndicator;第 2 组报告了最低的修改复杂性值 17.28.如图 11 所示,第 3 组报告了最高的构件影响复杂性值 12.624.在系统层面:第 1 组报告了最高的修改复杂性值 10 和影响复杂性值 12,并且具有最高的系统测试套更新复杂性值 146.6(表 7);第 2 组的系统修改和影响复杂性值是最小的,分别为 6.99 和 1.5,而且也具有最小的构件和系统测试套更新复杂性,其值分

别为 87.18 和 122.8.

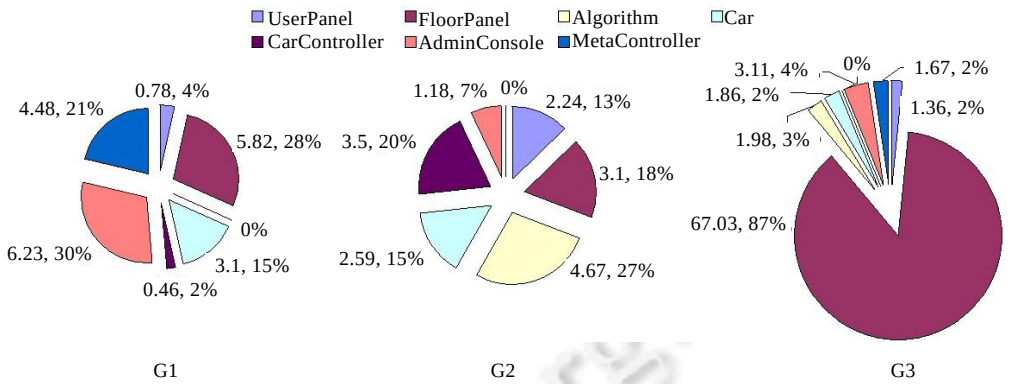



Fig.8 Component change complexity pie-chart

图 8 构件修改复杂性统计分布图

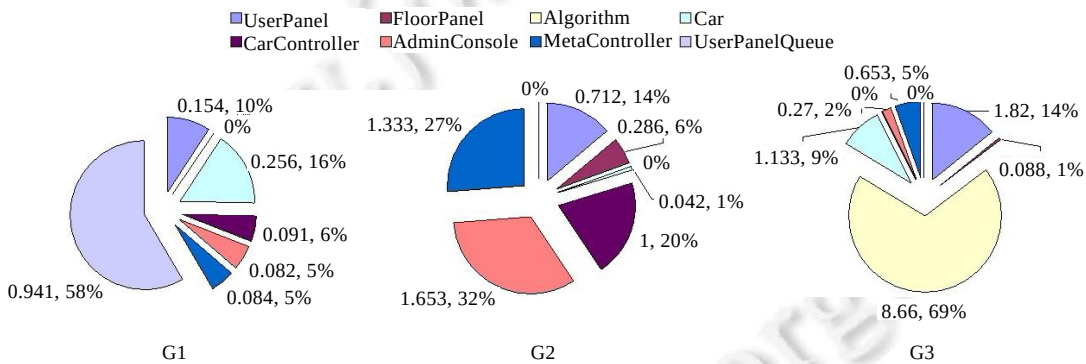



Fig.9 Component impact complexity pie-chart

图 9 构件影响复杂性统计分布图

**观察结论 3**. 对于某些构件修改,可能在构件层面引起的影响比较小,但会导致整个系统层面比较大的影响.

比如,第 1 组总的构件和系统复杂性是 20.87,总的影响复杂性是 1.61;而第 2 组相应的值为 17.28 和 5.03.也就是说:第 2 组的构件修改复杂性小于第 1 组,但其构件影响复杂性却大于第 1 组;不过,第 1 组的系统修改影响复杂性要大于第 2 组.这个实验结果也表明了:构件修改不仅仅会影响构件本身,也会对整个构件系统产生影响.

**观察结论 4**. 修改复杂性因素在构件层面的影响复杂性中起着重要作用.

如图 10 所示:第 1 组和第 2 组的构件修改复杂性分别为 20.87 和 17.28,但他们相应的构件影响复杂性分别是 1.61 和 5.03(图 11 所示).因此,第 1 组的构件修改复杂性要大于第 2 组,但构件影响复杂性却要低于第 2 组.根据实验中的调查分析,虽然第 1 组的构件内部数据和功能函数的修改数量要多于第 2 组,但第 2 组添加和修改了很多不同构件的 API,所以导致了比较高的影响复杂性.

**观察结论 5**. 在构件或者系统层面,测试套更新的复杂性是与总的修改和影响复杂性之和成正比的.

在构件层面,3 个实验组的修改和影响复杂性值之和分别是 22.49,22.21,89.63,而相对应的测试套更新复杂性值是 103.44,87.18,178.5.在系统层面,总的修改和影响复杂性值之和是 22,8.49,18.68,相应的测试套更新复杂

性值是 146.4,122.8,131.5.总的来说,测试套更新复杂性与总的修改和影响复杂性是成正比的.

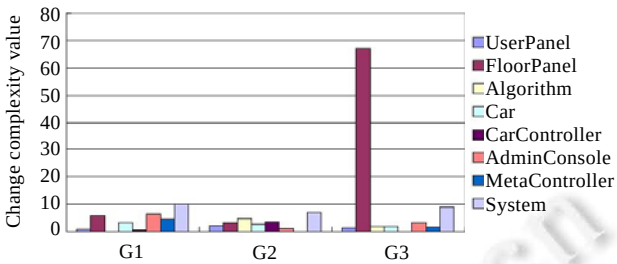



Fig.10 Summary of change complexity for each group

图 10 每组实验的构件及系统修改复杂性统计

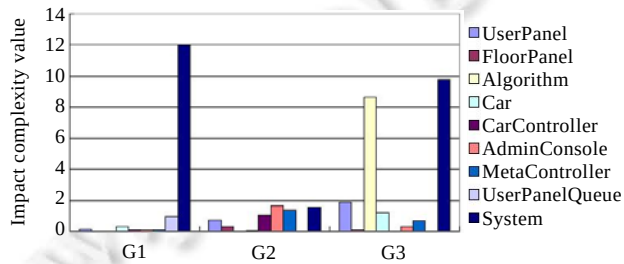



Fig.11 Summary of impact complexity for each group

图 11 每组实验的构件及系统影响复杂性统计

通过实验结果数据分析得出结论:不同的修改类型和复杂性因素会带来不同的影响及测试套更新,因此会导致不同的回归测试复杂性因素.比如说第 3 组添加了一个新的构件 FloorIndicator,带来了比较高的修改复杂性.所以在实际应用中,为了减少回归测试的复杂性,我们应该尽量避免添加新的构件.

总之,修改因素在修改和影响的复杂性中起着重要作用,有些因素带来比较多的复杂性,有些可能影响比较小.不同的修改构件也会导致不同的影响,通常情况下,核心构件或者复杂构件的修改会带来更高的复杂性.另外,构件修改会给构件层面和系统层面都带来影响,比如在实验中,API、消息和端口的修改会给系统带来比较明显的影响.测试套更新复杂性是与修改以及影响复杂性相关的,比如系统层面的修改和影响会给构件和系统层面的测试复杂性带来很大影响.

### 3.5 对实验结果有效性的威胁和不足

我们的实验中还存在以下一些对实验结果有效性的威胁和不足:

防火墙的方法在识别修改影响分析中还是略显粗糙,不可能将所有的影响都包含在防火墙中.也就是说,在防火墙外部也可能存在一些由于修改所带来的影响或者程序错误.因此,安全性问题有待进一步考证.我们也没有考虑 GUI 修改、外部环境或者硬件的变化.这些都有可能对测试复杂性产生影响.另外,我们实验的对象程序主要用于学术使用,在复杂的工业环境中可能面临更多的实际情况;

实验本身的参与者是研究生.以往研究经验也表明了实验结果在推广到实际工业软件维护中的局限性.但在当前软件工程学术研究中的可控制实验一般都是由学生所完成,因为很多实验条件和环境是在工业中是很难复制和控制的.我们提出的回归测试复杂性度量框架可能存在一些复杂性因素没有考虑,比如开发者或者修改者的认知复杂性、人工分析复杂性等.另外,要进一步表明方法的有效性和效率,还需要更多的实验验证.

## 4 相关工作

### 4.1 回归测试相关的复杂性分析

复杂性度量可以用来预测软件系统的可维护性和可依赖性.另外,复杂性度量还可以在软件开发和维护过程中提供反馈意见来控制流程.度量的主要应用在于预测、质量控制、产品评估以及工程控制.目前有很多不同的软件复杂性度量方法,比如 McCabe 圈复杂性、McClure 控制流度量、面向功能的度量、代码行度量等[14].在软件维护中的复杂性度量相关研究主要集中在:通过复杂性度量来进行软件维护和演化的错误及成本预测.

Kemerer 等人在文献[15]中总结了复杂性度量在软件维护过程中的应用,发现软件维护的复杂性主要与软件修改、程序错误及成本相关.Hassan 讨论了基于代码修改过程的复杂性度量,包括错误修复修改、通常维护修改、属性引入修改.他们通过一些修改模型,使用历史代码修改信息来定量分析维护的复杂性[6].实验研究表明,度量修改复杂性可以比其他度量元素更好地预测程序的潜在错误.Kafura 和 Reddy 总共使用 7 种代码度量和结构度量手段对实际系统连续的版本进行实验分析[16],实验结果表明了所提出的度量方式可以有效地识别维护的复杂性.Nikora 和 Munson 提出了一种针对软件演化的复杂性度量方法[7],他们主要通过控制流图中的节点和边的度量来预测植入到系统中的错误.研究表明:系统结构化度量可以有效预测植入系统中的错误个数,还能发现不同的修改会导致不同的错误类型.

### 4.2 回归测试成本分析模型和评估

迄今为止,研究者们已经提出了大量回归测试用例选择技术,涉及的研究范围也很广.那么,如何评价这些回归测试用例选择的有效性,满足成本和效益之间的均衡,则需要进行成本有效性分析.目前,对回归测试选择技术成本模型的研究主要有两方面:成本评估模型和成本预测模型.评估模型主要分析影响成本的各种因素,从中提取计算成本的模型,可以用来评估各种回归测试选择技术的成本有效性.预测模型是通过某种度量方式,比如测试用例覆盖率、测试复杂性等,来预测回归测试选择的成本有效性.

Leung 和 White 等人分析了可能影响回归测试成本的各种因素,其中包括系统分析成本、测试用例选择成本、测试用例执行成本和测试结果分析成本,然后给出了一种简单的成本评估模型(LW 模型)[13].回归测试用例选择技术是否具有成本有效性,通常需要满足条件:选择成本和选择出的测试用例执行成本之和少于重新运行所有测试用例的成本.Rosenblum 等人基于 Leung 和 White 提出的回归测试用例选择成本模型,结合给定的一些假设的基本条件,提出了一种能够确定回归测试选择成本有效性的预测器[17].根据他们的经验,在软件维护阶段,测试用例及被其覆盖的程序部分这种覆盖关系一般比较稳定,很少变化,除非有大型子系统或属性被添加到程序中.如果在不同的版本之间这种变化很小的话,那么可以认为这种关系是基本稳定的.同时也发现:回归测试选择技术排除冗余测试用例的能力,主要是由这种覆盖关系所决定的.Harrold 等人在该预测器的基础上,通过实验进一步分析指出:不仅代码覆盖会影响成本有效性,而且程序修改的位置也会对预测的精确性产生影响[18].于是,他们把修改信息添加到预测器中,通过对修改频度加权,拓展了 Rosenblum 等人提出的方法.Malishevsky 等人为回归测试选择技术、优先级技术等建立了成本效益模型[19],并认为 Leung 和 White 等人的成本模型对于安全性回归测试选择比较合适;但对于不安全的回归测试选择,LW 模型却没有考虑由于丢弃测试用例而导致的错误遗漏成本.Malishevsky 等人在 LW 模型的基础上重新考虑了几种影响回归测试成本的变量因素,其中包括分析成本、执行成本、结果检测成本、选择成本、测试集维护成本和错误忽略成本.Rothermel 等人还研究了测试集粒度对回归测试成本有效性的影响[20].

软件测试的任务是寻找程序错误,回归测试也是以找错误为目标.如果某种测试用例选择技术选择了所有的能够揭示修改后程序的错误的测试用例,那么该方法就是安全的.换言之,就是该回归测试选择方法不遗漏任何能够揭示错误的测试用例.Rothermel 和 Harrold 对被重用的回归测试用例进行了划分[21]:如果某个测试用例使得修改后的程序不能通过测试,则该测试用例是错误揭示的(fault-revealing);如果某个测试用例使得修改后程序和原程序产生不同的输出,那么该测试用例是修改揭示的(modification-revealing);如果某个测试用例执行了修改后程序添加、修改或被删除的代码,那么该测试用例是修改遍历的(modification-traversing).修改遍历的

测试用例集包含修改揭示的测试用例集,而修改揭示的测试用例集包含错误揭示的测试用例集.在实际回归测试中,如果要准确地找出这 3 种测试用例都是比较困难的.如果某种测试用例选择技术选择了所有的修改揭示测试用例,那么可以认为该方法是安全的.当然,如果选择了修改揭示测试用例集的超集——修改遍历的测试用例集,那么该方法也是安全的.当然,也可以根据修改-遍历测试用例或者其他测试选择标准,比如数据流测试选择等来重新定义相关度量.Gallagher 等人认为:包含性主要是从安全性的角度定义的,而精确性主要反映的是排除不相关测试用例的能力[22].他们对 Rothermel 等人的评估框架进行了局部改动,用安全性(safety)来度量包含性,把精确性改为排斥性(exclusiveness).他们给出的 4 种评估是:安全性、排斥性、有效性和通用性,这样的改动更有利于对包含性和排斥性进行度量.防火墙技术的通用性和有效性都很好,而切片、数据流等技术由于复杂性较高,因而有效性受到很大影响.

## 5 结束语和展望

当前的一些研究工作主要集中在软件维护过程的复杂性度量和回归测试的成本模型分析,但缺乏从软件维护的角度来研究构件软件回归测试的复杂性度量问题.有效的回归测试复杂性度量,包括修改、影响及测试套更新,可以帮助测试管理者和软件质量保证人员管理维护和重测过程和制定策略.我们的方法提供了一个系统化的度量框架,包括复杂性度量模型和形式化度量方法,用来解决软件的回归测试复杂性分析和度量问题.我们从研究目标、方法和模型以及方法结果这几个方面总结了本文的研究贡献如下:

- 研究目标

现有的相关工作主要研究软件维护或演化的代码度量或者结构度量[6,15,16],而针对回归测试的研究主要是成本-效益(cost-benefit)模型[13,17−18].与已有的工作不一样,我们针对回归测试的复杂性,而这种复杂性是在软件维护过程中的,包括修改、影响以及测试套的更新,而且是以常见的构件软件为研究对象.

- 方法和模型

已有的工作主要使用 McCabes 圈复杂性、McClures 控制流、面向功能度量或者代码行的度量这些方法[14],大部分复杂性度量模型都是结构化的或者基于代码的[6,16],他们没有考虑软件修改及修改带来的影响中所存在的不同复杂性因素以及这些因素在回归测试复杂性度量中的应用.我们从构件和系统两个层面分析并总结了构件软件中的不同复杂性因素,根据复杂性因素,提出了一种基于模型的回归测试复杂性度量框架,包括复杂性因素的度量以及多因素回归测试复杂性.相比于传统的线性复杂性计算,该模型可以对发生修改的构件或者系统的修改及重测的影响复杂性进行可视化的表示和比较.另外,通过复杂性比较可以发现不同回归测试手段的复杂性差异,从而有利于软件维护人员选择合适的修改及回归测试方法.

- 方法结果

目前,还没有针对构件软件回归测试复杂性研究的实验分析.我们通过真实的构件系统执行完整的回归测试过程,然后利用本文提出的度量框架进行复杂性分析.通过实验,我们得到了一些观察结果和启示.实验结果也表明,我们提出的方法是可行的和有效的.

在未来的研究工作中,我们考虑将复杂性度量和测试成本建立关联,研究如何进行构件软件回归测试的成本评估以及使用经验模型来预测回归测试成本.

**References**:

[1] Gao J, Gopinathan D, Mai Q, He JS. A systematic regression testing method and tool for software components. In: Proc. of the 30th Annual Int'l Computer Software and Applications Conf. 2006. 455−456. [doi: 10.1109/COMPSAC.2006.17]

[2] Gao J, Chen C, Toyoshima Y, Kung DC, Hsia P. Identifying polymorphism change and Impact in Object-orientated Software Maintenance. Journal of Software Maintenance: Research and Practice, 1996,8(6):357−387. [doi: 10.1002/(SICI)1096-908X (199611)8:6<357::AID-SMR139>3.0.CO;2-S]

[3] Orso A, Harrold MJ, Rosenblum D, Rothermel G. Using component metacontents to support the regression testing of component-based software. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. 2001. 716−725. [doi: 10.1109/ICSM.2001.972790]

[4] Tao CQ, Li BX, Gao J, Sun XB. Model-Based change impact analysis for component-based software. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):943−960 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4371.htm [doi: 10.3724/SP.J.1001.2013.04371]

[5] Tao CQ, Li BX, Gao J. A model-based approach to regression testing of component-based software. In: Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering. 2011. 230−237.

[6] Hassan AE. Predicting faults using the complexity of code changes. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 78−88. [doi: 10.1109/ICSE.2009.5070510]

[7] Nikora AP, Munson JC. An approach to the measurement of software evolution. Journal of Software Maintenance and Evolution: Research and Practice, 2005,17(1):65−91. [doi: 10.1002/smr.303]

[8] Kaoru I. Introduction to quality control. ISBN 4-906224-61-X OCLC 61341428, 1990. 448.

[9] Fournier A, Montuno DY. Triangulating simple polygons and equivalent problems. ACM Trans. on Graphics, 1990,3(2):153−174. [doi: 10.1145/357337.357341]

[10] Ahn Y, Suh J, Kim S, Kim H. The software maintenance project effort estimation model based on function points. Journal of Software Maintenance and Evolution: Research And Practice, 2003,15(2):71−85. [doi: 10.1002/smr.269]

[11] Li W. QoS assurance for dynamic reconfiguration of component-based software. IEEE Trans. on Software Engineering, 2012,38(3): 658−676. [doi: 10.1109/TSE.2011.37]

[12] Tai KC. Program testing complexity and test criteria. IEEE Trans. on Software Engineering, 1980,6(6):531−538. [doi: 10.1109/TSE.1980.234501]

[13] Leung HKN, White LJ. A cost model to compare regression test strategies. In: Proc. of the Int'l Conf. on Software Maintenance. 1991. 201−208. [doi: 10.1109/ICSM.1991.160330]

[14] Pressman RS. Software Engineering: A Practitioner's Approach. Mc Graw Hill, 2008.

[15] Kemerer CF. Software complexity and software maintenance: A survey of empirical research. Annals of Software Engineering, 1995,1(1):1−22. [doi: 10.1007/BF02249043]

[16] Kafura D, Reddy GR. The use of software complexity metrics in software maintenance. IEEE Trans. on Software Engineering, 1987,13(3):335−343. [doi: 10.1109/TSE.1987.233164]

[17] Rosenblum D, Weyuker E. Using coverage information to predict the cost-effectiveness of regression testing strategies. IEEE Trans. on Software Engineering, 1997,23(3):146−156. [doi: 10.1109/32.585502]

[18] Harrold MJ, Rosenblum D, Rothermel G, Weyuker E. Empirical studies of a prediction model for regression test selection. IEEE Trans. on Software Engineering, 2001,27(3):248−263. [doi: 10.1109/32.910860]

[19] Malishevsky A, Rothermel G, Elbaum S. Modeling the cost-benefits tradeoffs for regression testing techniques. In: Proc. of the Int'l Conf. on Software Maintenance. 2002. 204−213. [doi: 10.1109/ICSM.2002.1167767]

[20] Rothermel G, Elbaum S, Malishevsky A, Kallakuri P, Davia B. The impact of test suite granularity on the cost-effectiveness of regression testing. In: Proc. of the Int'l Conf. on Software Engineering. 2002. 19−25. [doi: 10.1145/581339.581358]

[21] Rothermel G and Harrold MJ. Analyzing regression test selection techniques. IEEE Trans. on Software Engineering, 1996,22(8): 529−551. [doi: /10.1109/32.536955]

[22] Gallagher K, Hall T, Black S. Reducing regression test size by exclusion. In: Proc. of the Int'l Conf. on Software Maintenance. 2007. 154−163. [doi: 10.1109/ICSM.2007.4362628]

## 附中文参考文献:

[4] 陶传奇,李必信,Gao J,孙小兵.基于模型的构件软件修改影响分析.软件学报,2013,24(5):943−960. http://www.jos.org.cn/1000-9825/4371.htm [doi: 10.3724/SP.J.1001.2013.04371]

**陶传奇**(1984－),男,安徽安庆人,博士,讲师,CCF 会员,主要研究领域为软件测试,软件维护.

**李必信**(1969－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为软件建模、分析、测试与验证,软件维护相关技术.

**Jerry Gao**(1960－),男,博士,教授,博士生导师,主要研究领域为面向对象软件测试,构件软件测试,云测试.