对象驱动的软件测试性度量

付剑平 刘 斌

(北京航空航天大学 可靠性与系统工程学院, 北京 100191)

摘 要:为了使用统一的方式度量各种软件的测试性,降低测试性度量的难度和费用,提出一种对象驱动的软件测试性度量方法. 方法基于一个由测试性度量、易测试特性度量、测试性因素度量及其关系构成的软件测试性度量框架,首先分析度量的对象,确定对象的类型、对象所属软件的类型和测试情况;然后根据软件的类型和测试情况,从度量框架中选取匹配的元素,构造一个专用的软件测试性度量框架;最后根据度量对象的类型,使用专用软件测试性度量框架中的元素计算测试性因素、易测试特性或测试性. 应用表明方法能够得到软件测试性的定量结果,预计测试资源,为软件测试提供帮助.

关 键 词: 软件测试; 测试性; 度量; 对象驱动

中图分类号: TP 311.5

文献标识码: A **文 章 编 号**: 1001-5965(2011)12-1569-06

Object-driving software testability measurement

Fu Jianping Liu Bin

(School of Reliability and Systems Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: To reduce the difficulty and cost of software testability measurement, an object-driving method was proposed to measure the testability of various programs by a unified manner. It is based on a software testability metrics framework that consists of the testability metric, testable characteristics metrics, testability factors metrics and their relations. First, the measurement object was analyzed to determine the object type, the software type and test conditions. Then suited elements were chose from the metrics framework according to the software type and test conditions to construct a special software testability metrics framework. Finally advantage of the elements in the special metrics framework was taken to calculate the testability factors, testable characteristics or testability according to the object type. The application indicates this method can get the quantitative value of software testability and help the software testing by predicting the test resources.

Key words: software testing; testability; measurements; object driving

软件测试性能够反映对一个软件进行测试的难易程度,了解软件测试性,能对软件的测试、设计和发布提供帮助. 测试人员能够根据测试性的大小预计测试资源、安排测试,设计人员能够针对测试性的薄弱环节改进设计、提高测试性,项目管理人员能够根据测试性判断软件测试是否已经足够、能否发布软件. 这些过程中的测试性信息都由测试性分析得到,测试性度量则为测试性分析提供定量结果.

度量是从模型或理论中抽取规则、并根据规则为对象分配数值的活动,其中的 3 个要素:对象、规则和数值,又是方法好坏的判别标准. 好的度量方法能够通过更简单的规则,度量更多的对象,得到更准确的数值. 研究可知,目前的软件测试性度量方法只能度量有限的对象,如区域测试性方法[1] 只能度量构件的可观察性和可控性,

收稿日期: 2010-08-12; **网络出版时间**: 2011-12-21 09:57; **DOI**: CNKI:11-2625/V.20111221.0957.012
网络出版地址: www.cnki.net/kcms/detail/11.2625.V.20111221.0957.012.html

作者简介: 付剑平(1979 -),男,湖南耒阳人,博士后,fjp@buaa.edu.cn.

PIE(Propagation, Infection, Execution)技术[2-5]只能计算程序的敏感性,测试性度量框架[6]只能分析面向对象软件的测试性因素,验证度量方法[7]只能得到需求和设计的测试性,宏观测试性模型[8]只适用于进行随机黑盒测试的大型程序,文献[9]也只解决了并发程序的测试性分析问题.这就使得软件在开发过程中为了获取不同的测试性信息必须使用不同的度量方法;同样,对于具有不同特征的软件也需使用不同的度量方法.由于每种方法在度量过程、计算模型上都存在差异,使用多种方法的难度和费用显然要高于仅使用一种方法.

为了降低软件测试性度量的难度和费用,本文提出了一种对象驱动的软件测试性度量方法.方法能根据对象的不同选择不同的元素,采用相同的度量过程和计算模型确定对象的数值.

1 软件测试性度量对象

软件测试性度量主要为 3 类对象分配数值:测试性、易测试特性和测试性因素.测试性是系统、组件或需求为建立测试准则及执行测试提供便利的程度[10],也有人认为测试性是在某一特定的输入分布下软件中包含的一个缺陷在下一次测试执行过程中导致失效的概率[2].易测试特性是指那些与测试性相关的软件特性,分别代表了测试性的一个方面,常见的易测试性特性有可理解性、可观察性、可控性等[7].测试性因素是指那些能影响测试性或易测试特性的软件属性,如代码行数、圈复杂度和遗传树深度[11].

由于不同类型的软件具有其独特的测试性因素,如遗传树深度就是面向对象软件的测试性因素而非结构化软件的测试性因素,这些测试性因素又会影响到易测试特性和测试性,因此软件测试性度量会受到软件类型的影响,确定度量对象的过程中需要明确对象所属的软件的类型.软件类型的划分方式很多,本文主要考虑软件的开发方法和应用领域.

软件测试性是与软件测试相关的软件属性,软件在不同的测试情况下会表现出不同的测试性,例如一个在单元测试时表现良好的软件,系统测试时的测试性有可能很差,为此在确定度量对象时还需要明确对象所属的软件的测试情况,包括测试的层次、类型、方法等.

综上,为了支持后续度量,分析软件测试性度量对象时除了需要确定对象的类型,还应确定对象所属的软件的类型和测试情况.

$$o = <t_{obj}, t_{SW}, C_{test}> \quad (1)$$

式(1)中,o 表示软件测试性度量对象,它包括 3 类特性:t_{obj},t_{SW} 和 C_{test};t_{obj} 表示对象类型,$t_{obj} \in$ {测试性,易测试特性, 测试性因素};t_{SW} 表示对象所属软件的类型;C_{test} 表示对象所属软件的测试情况,其中

$$t_{SW} = <m_{dev}, f_{app}> \quad (2)$$

$$C_{test} = <L_{test}, T_{test}, M_{test}> \quad (3)$$

式(2)中,m_{dev} 表示对象所属软件的开发方法,$m_{dev} \in$ {结构化方法, 面向对象方法, 基于构件方法, …};f_{app} 表示对象所属软件的应用领域,$f_{app} \in$ {文字处理, 实时控制, 科学计算, …}.式(3)中,L_{test} 表示对象所属软件需要进行哪些级别的测试,$L_{test} \subset$ {单元测试, 集成测试, 系统测试};T_{test} 表示对象所属软件需要进行哪些类型的测试,$T_{test} \subset$ {功能测试, 性能测试, 接口测试, …};M_{test} 表示对象所属软件在测试过程中需要使用哪些测试方法,$M_{test} \subset$ {等价类划分, 边界值分析, 状态测试, …}.

2 软件测试性度量框架

如前所述,不同软件的测试性度量对象都有其不同的特性,这些特性会影响到软件的测试性度量.为了能用同样的方式度量软件,就必须通过特性得到软件的一些共性元素,然后基于共性元素进行计算.软件测试性、易测试特性以及测试性因素是任何软件都具备的软件属性,软件测试性包含易测试特性,易测试特性受测试性因素影响也是这 3 类软件属性之间的固有关系.测试性因素是最基本的度量单元,其计算方式可根据定义得到;测试性因素决定易测试特性,因此易测试特性可通过对测试性因素的综合评价得到;同样,测试性也能通过其包含的易测试特性综合得到.因此,无论何种软件只要确定了测试性、易测试特性、测试性因素以及它们之间的关系,就能通过同样的方式进行计算.

为了能用同样的方式得到软件的共性元素,本文将各种软件的测试性、易测试特性、测试性因素度量以及它们的关系组织起来,结合软件的类型和测试信息共同组成一个软件测试性度量框架 F_{met}:

$$F_{met} = <m_T, M_{cha}, M_{fac}, R_{inc}, R_{aff}> \quad (4)$$

式中,m_T 表示软件测试性的一个度量;M_{cha} 表示一个由易测试特性的度量组成的集合;M_{fac} 表示一个由测试性因素的度量组成的集合;R_{inc} 表示测试性度量与易测试特性度量之间的关系构成的集

合;R_{aff}表示易测试特性度量与测试性因素度量之间的关系构成的集合.

由于软件测试性尚未形成统一定义,软件测试性包含哪些易测试特性也未达成共识,F_{met}使用表 1 定义的测试性和易测试特性度量.

由表 1 可知,F_{met}中 m_T = ST 并且 M_{cha} = {Und, Con, Obv, TSC, Sim, Dec, App, Tra, Sen}. M_{fac}中任一测试性因素度量 m_{fac}的表示如下:

$$m_{fac} = < n_{fac},\ d_{fac},\ R_{fac} > \quad (5)$$

其中,n_{fac}表示 m_{fac}的名称;d_{fac}表示 m_{fac}的定义;R_{fac}表示 m_{fac}的适用范围. R_{fac}包括 5 个属性:开发方法(M_{dev})、应用领域(F_{app})、测试级别(L_{test})、测试类型(T_{test})和测试方法(M_{test}),指明了 m_{fac}能用于哪些软件在哪些测试情况下的测试性度量,每个属性都有一个取值集合,取值为"Null"表明 m_{fac}在该属性上无限制,表 2 为 F_{met}中的部分测试性因素度量.

R_{inc}和 R_{aff}中的任一关系 r 也能用源和目的组成的二元组表示:

$$r = < s_r,\ d_r > \quad (6)$$

对于 $r \in R_{inc}$, s_r = ST 并且 $d_r \in M_{cha}$,表明软件测试性包含易测试特性,同时 R_{inc}中的元素个数等于 M_{cha}中的元素个数;对于 $r \in R_{aff}$, $s_r \in M_{fac}$并且 $d_r \in M_{cha}$,表明测试性因素影响易测试特性.

表 1　测试性及易测试特性度量

软件特性	度量	定义
软件测试性	ST	对软件进行测试、发现软件缺陷的难易程度
可理解性	Und	为更好地理解软件,软件提供信息的表达程度
可控性	Con	软件为施加外部输入、控制内部状态提供便利的程度
可观察性	Obv	软件为观察外部输出、监控内部状态提供便利的程度
测试支持能力	TSC	软件为各种测试工具的使用提供便利的程度
简单性	Sim	在满足需求的基础上软件功能、结构的简洁程度
可分解性	Dec	软件能被分解为独立的模块进行单独测试的概率
适用性	App	软件能在各种使用环境下正常运行的概率
可跟踪性	Tra	软件为跟踪自身功能操作、属性和行为提供便利的程度
敏感性	Sen	软件在测试过程中发生失效的概率

表 2　测试性因素度量

n_{fac}	d_{fac}	R_{fac}				
		M_{dev}	F_{app}	L_{test}	T_{test}	M_{test}
NEO	软件外部输出数目	结构化方法 面向对象方法	科学计算 实时控制	系统测试	接口测试	Null
MCC	软件在工作和测试模式之间相互切换的能力	结构化方法 面向对象方法	科学计算 实时控制	单元测试 集成测试 系统测试	Null	Null
OP_{min}	总线输出数据最大延迟周期的最小值	结构化方法	科学计算 实时控制	系统测试	性能测试 余量测试	Null

3　对象驱动的度量过程

基于软件测试性度量框架 F_{met},对象驱动的软件测试性度量过程如下:

1) 分析度量对象 o,确定其 t_{obj}, t_{SW}和 C_{test}信息.

2) 根据 t_{SW}和 C_{test}信息,从 F_{met}中选择能用于对象 o 软件测试性度量的元素,过程如下:

① 抽取测试性因素度量. 对 $\forall m_{fac} \in M_{fac}$,如果 m_{fac}的适用范围能够覆盖对象 o 所属软件的类型和测试信息,就将 m_{fac}纳入对象 o 的测试性因素度量集合 M^o_{fac},范围覆盖的判断准则为:

$((m_{fac}.R_{fac}.M_{dev} \ni o.t_{SW}.m_{dev} \vee m_{fac}.R_{fac}.M_{dev} = \text{Null}) \wedge (m_{fac}.R_{fac}.F_{app} \ni o.t_{SW}.f_{app} \vee m_{fac}.R_{fac}.F_{app} = \text{Null}) \wedge (m_{fac}.R_{fac}.L_{test} \cap o.C_{test}.L_{test} \neq \phi \vee m_{fac}.R_{fac}.L_{test} = \text{Null}) \wedge (m_{fac}.R_{fac}.T_{test} \cap o.C_{test}.T_{test} \neq \phi \vee m_{fac}.R_{fac}.T_{test} = \text{Null}) \wedge (m_{fac}.R_{fac}.M_{test} \cap o.C_{test}.M_{test} \neq \phi \vee m_{fac}.R_{fac}.M_{test} = \text{Null})) = \text{True}$

② 抽取易测试特性与测试性因素度量之间的关系. 对 $\forall r \in R_{aff}$,如果 $r.s_r \in M^o_{fac}$,则将 r 纳入对象 o 的易测试特性与测试性因素度量之间的关系集合 R^o_{aff};

③ 抽取易测试特性度量. 对 $\forall r \in R^o_{aff}$,将 $r.d_r$ 纳入对象 o 的易测试特性度量集合 M^o_{cha};

④ 抽取测试性与易测试特性度量之间的关系. 对 $\forall r \in R_{inc}$,如果 $r.d_r \in M^o_{cha}$,则将 r 纳入对象 o 的测试性与易测试特性度量关系集合 R^o_{inc};

⑤ 组合上述过程得到的 M^o_{fac}, R^o_{aff}, M^o_{cha}, R^o_{inc},结合 ST 得到一个专用于对象 o 的软件测试性度

量框架 $F^o_{\text{met}} = <\text{ST}, M^o_{\text{cha}}, M^o_{\text{fac}}, R^o_{\text{inc}}, R^o_{\text{aff}}>$.

3）根据 t_{obj} 信息，确定 F^o_{met} 中各度量的数值：

① 如果 t_{obj} 为“测试性因素”，对 $\forall m_{\text{fac}} \in M^o_{\text{fac}}$，根据 m_{fac} 的定义分析对象 o 所属软件的需求、设计和实现，得到 m_{fac} 的取值；

② 如果 t_{obj} 为“易测试特性”，首先确定 M^o_{fac} 中各测试性因素度量的取值，然后对 $\forall m_{\text{cha}} \in M^o_{\text{cha}}$，分析与 m_{cha} 相关的测试性因素度量，综合这些度量计算 m_{cha} 的取值，计算方法如文献[12]所述；

③ 如果 t_{obj} 为“测试性”，则首先确定 M^o_{fac} 中各测试性因素度量的取值，然后计算 M^o_{cha} 中各易测试特性的度量，最后采用文献[12]中的模型计算测试性.

4 软件测试性度量实例

以某型飞机的任务计算机(MC，Mission Computer)软件为例，阐明如何应用对象驱动的方法进行软件测试性度量并根据度量结果预计测试资源.

1）MC 软件要求进行系统测试，重点包括功能测试、性能测试、强度测试、接口测试和边界测试，预计使用等价类划分、边界值分析和状态测试方法. 为了预计测试资源、制定测试计划，需要度量软件的测试性. 确定度量对象 o_{MC} 的基本信息见表 3.

表 3 o_{MC} 基本信息

t_{obj}	t_{SW}		C_{test}		
	m_{dev}	f_{app}	L_{test}	T_{test}	M_{test}
测试性	结构化方法	实时控制	系统测试	功能、性能强度、接口边界测试	等价类划分边界值分析状态测试

2）逐个判断 M_{fac} 中的元素是否满足覆盖准则，以 NEO 为例，由于 NEO. R_{fac}. $M_{\text{dev}} \ni o_{\text{MC}}$. t_{SW}. m_{dev}，NEO. R_{fac}. $F_{\text{app}} \ni o_{\text{MC}}$. t_{SW}. f_{app}，NEO. R_{fac}. $L_{\text{test}} \cap o_{\text{MC}}$. C_{test}. $L_{\text{test}} \neq \phi$，NEO. R_{fac}. $T_{\text{test}} \cap o_{\text{MC}}$. C_{test}. $T_{\text{test}} \neq \phi$ 并且 NEO. R_{fac}. M_{test} = Null，可知 NEO 满足覆盖条件，因此将 NEO 纳入 o_{MC} 的测试性因素度量集合 $M^{\text{MC}}_{\text{fac}}$，最后得到 $M^{\text{MC}}_{\text{fac}} = \{$FPI, NNFR, KLOC, CORAD, CODAI, COR, COD, FRR, ERR, QRR, ARD, ARI, NEI, NEO, AEIDL, LOEIDS, NCS, SOR, SIR, NIT, IP_{min}, RC, MCC, ISCC, OP_{min}, ISOC, EOI$\}$，各度量的具体定义如表 4 所示.

表 4 $M^{\text{MC}}_{\text{fac}}$ 测试性因素度量定义

度量	定义
FPI	软件的功能点数
NNFR	软件的非功能需求数
KLOC	程序的千行代码数
CORAD	需求与设计的一致程度
CODAI	设计与实现的一致程度
COR	需求的一致程度
COD	设计的一致程度
FRR	可实现的需求占所有需求的比
ERR	具有预期的需求占所有需求的比
QRR	具有定量要求的需求占所有需求的比
ARD	需求描述的详细程度
ARI	需求描述的完整程度
NEI	软件外部输入数目
AEIDL	外部接口描述的详细程度
LOEIDS	外部输入依赖其他输入或输出的程度
NCS	与被测软件交联的其他系统数目
SOR	软件的标准输出接口与输出接口总数之比
SIR	软件的标准输入接口与输入接口总数之比
NIT	软件接口的类型数目
IP_{min}	总线输入数据最大延迟周期的最小值
RC	软件能否重置到初始状态的能力
ISCC	可控制的中间变量数目与中间变量总数之比
ISOC	可观察到的中间变量数目与中间变量总数之比
EOI	外部输出的直观程度

在 $M^{\text{MC}}_{\text{fac}}$ 的基础上依次确定易测试特性与测试性因素度量的关系集合 $R^{\text{MC}}_{\text{aff}}$、易测试特性度量集合 $M^{\text{MC}}_{\text{cha}}$ 以及测试性与易测试特性度量的关系集合 $R^{\text{MC}}_{\text{inc}}$，构造出 o_{MC} 的软件测试性度量框架 $F^{\text{MC}}_{\text{met}}$ 如图 1 所示，连线表示元素之间的关系.

3）由于 t_{obj} 为“测试性”，首先确定 $M^{\text{MC}}_{\text{fac}}$ 中各测试性因素度量的取值，见表 5.

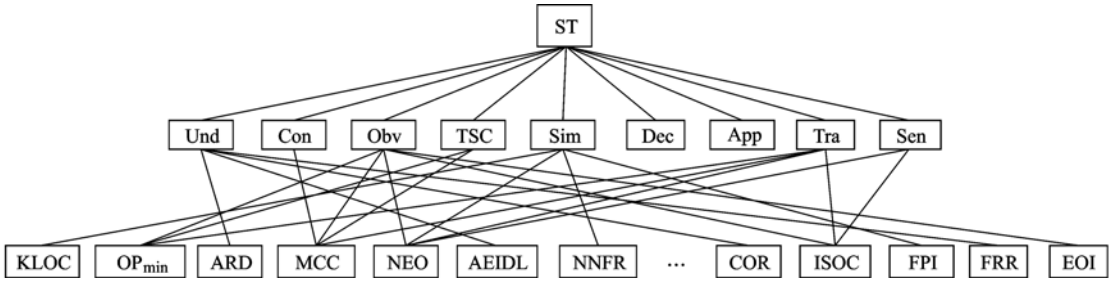

图 1 软件测试性度量框架 $F^{\text{MC}}_{\text{met}}$

表 5　MC 软件测试性因素度量值

度量	值	度量	值
FPI	48	QRR	0.508 5
NNFR	11	ARD	0.940 7
KLOC	31	ARI	0.627 1
CORAD	0.166 7	NEI	82
CODAI	0.125	NEO	78
COR	0.5	AEIDL	0.993 8
COD	0.25	LOEIDS	3.996
FRR	0.983 1	NCS	7
ERR	0.864 4	SOR	1
SIR	1	IP_{min}/ms	20
NIT	2	OP_{min}/ms	20
RC	1	ISOC	0.161 3
MCC	1	EOI	0.406 1
ISCC	0.096 8		

然后计算 M_{cha}^{MC} 中各易测试特性的度量,得到 |Und| = 0.903 1, |Con| = 0.694 2, |Obv| = 0.795 5, |TSC| = 0.987 5, |Sim| = 0.727 8, |Dec| = 0.555 6, |App| = 0.996 6, |Tra| = 0.643 3, |Sen| = 0.625 2; 最后确定 MC 软件的测试性 |ST| = 0.776.

4) 根据以往软件的测试性和测试资源数据,通过曲线拟合的方式得到它们之间的关系:

$$R = 60 - 587\ln T \tag{7}$$

其中,R 表示测试资源,人日;T 表示测试性. 将确定的 MC 软件的测试性代入式(7)得到预计的测试资源为 209 人日,在此基础上制定 MC 软件的测试计划.

5　方法分析

本文分析了各种常见的软件测试性度量方法,如表 6 所示,结果表明这些方法大都只能度量某类对象,如控制流度量方法只能度量软件测试性、区域测试性度量方法只能度量易测试特性等,单独使用一种方法无法满足各种软件测试性度量对象的度量需要. 不仅如此,各种软件测试性度量方法适用的软件类型和测试情况也有限,除了宏观测试性模型,其他方法只能用于结构化或面向对象方法开发的软件;除了面向对象度量框架,没有方法能度量软件在各个测试级别的测试性.

表 6　软件测试性度量方法分析

方法	度量对象							
	对象类型			开发方法		测试级别		
	测试性因素	易测试特性	测试性	结构化	面向对象	单元测试	集成测试	系统测试
区域测试性度量[1]		√		√		√		
动态 PIE 分析[2]			√	√		√		√
静态 PIE 分析[3]			√	√		√		√
变异 PIE 分析[4]			√		√	√		√
扩展 PIE 分析[5]			√	√		√		√
面向对象度量框架[6]	√				√	√	√	√
验证度量模型[7]		√	√		√			√
宏观测试性模型[8]			√	√	√	√		√
并发程序度量[9]	√		√	√				√
控制流度量[13]			√	√		√		
DDR(Domain-Range Ratio)度量[14]			√	√		√		
数据流度量[15]			√	√		√		
信息转移图度量[16]		√		√		√		
运行测试性度量[17]	√		√		√		√	
平台测试性评估[18]			√	√		√		
BPEL(Business Process Execution Language)测试性评估[19]		√			√			√
结构化程序测试性模型[20]	√			√				√

注:"√"表示方法能够度量该类型的对象,对象所属软件由该方法开发、并需进行该项级别的软件测试.

软件测试性是软件的一个质量特性,质量特性既包含内部属性也包含外部属性,内部属性在不同的使用环境下会对外部属性产生影响,从而影响到外部属性的度量[21]. 基于该原理,对象驱动的软件测试性度量方法从测试性因素(内部属性)出发,考虑软件在不同使用环境(应用领域、测试级别、测试类型、测试方法)下测试性因素对易测试特性(外部属性)的影响,计算易测试特性,最后获得软件测试性,因此适用于各种软件的测试性度量. 不仅如此,由度量过程可知,对象驱动的软件测试性度量方法还能得到各类定量结果,满足了不同对象的度量需要.

对象驱动的软件测试性度量需要从框架中选择与对象特性相符合的元素,因此完备的软件测试性度量框架是方法实施的关键. 完备的软件测试性度量框架不可能一蹴而就,需要工程实践的

不断积累.

6 结束语

本文提出了对象驱动的软件测试性度量方法,方法能根据软件测试性度量对象的不同特性,从软件测试性度量框架中选取不同的元素进行软件测试性度量,通过对实际软件的测试性度量证明了方法的可行性. 方法解决了目前软件测试性度量的难题,既不用为了获取各种软件测试性信息而使用多种度量方法,也不用因为软件类型或测试情况的变化而使用不同的方法,因此能够降低软件测试性度量的难度和费用,进而促进软件测试性度量的应用和发展.

参考文献(References)

[1] Freedman R S. Testability of software components [J]. IEEE Transactions on Software Engineering, 1991, 17(6): 553 – 564

[2] Voas J M, Miller K W. Software testability: the new verification [J]. IEEE Software, 1995, 12(3): 17 – 28

[3] Lin J C, Lin S W. An analytic software testability model [C]// Proc of 11th Asian Test Symposium. Guam: IEEE, 2002: 278 – 283

[4] Kansomkeat Supaporn, Offutt Jeff, Rivepiboon Wanchai. Class-component testability analysis[C]//Proc of the 5th WSEAS int Conf on Software Engineering. Madrid: IEEE, 2006: 61 – 66

[5] Tsai T H, Huang C Y, Chang J R. A study of applying extended PIE technique to software testability analysis [C]//Proc of 33rd Annual IEEE International Computer Software and Applications Conference. Seattle: IEEE, 2009: 89 – 98

[6] Mouchawrab Samar, Briand L C, Labiche Yvan. A measurement framework for object-oriented software testability [J]. Information and Software Technology, 2005, 47(1): 979 – 997

[7] Shih M C. Verification and measurement of software component testability [D]. San Jose: Department of Computer Engijeering, San Jose State University, 2004

[8] 吕俊廷. 软件的可测试性模型、度量及其应用[D]. 南京: 南京航空航天大学信息科学与技术学院, 2006
Lü Junting. Software testability model, metric and its application [D]. Nanjing: College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, 2006 (in Chinese)

[9] 陈振强, 徐宝文, 许蕾, 等. 一种并发程序可测试性分析框架[J]. 计算机学报, 2003, 26(12): 1685 – 1689
Chen Zhenqiang, Xu Baowen, Xu Lei, et al. A framework to analyze testability of concurrent programs [J]. Chinese Journal of Computers, 2003, 26(12): 1685 – 1689 (in Chinese)

[10] ANSI/IEEE Standard 610.12-1990 IEEE standard glossary of software engineering terminology[S]

[11] Bruntink M, Deursen A V. An empirical study into class testability [J]. The Journal of Systems and Software, 2006, 79(1): 1219 – 1232

[12] 付剑平, 陆民燕. 基于模糊综合评价的软件测试性度量[J]. 计算机工程与应用, 2009, 45(27): 69 – 71, 122
Fu Jianping, Lu Minyan. Software testability measurement based on fuzzy comprehensive evaluation [J]. Computer Engineering and Applications, 2009, 45(27): 69 – 71, 122 (in Chinese)

[13] Bache R, Mullerburg M. Measure of testability as a basis for quality assurance [J]. Software Engineering Journal, 1990, 5(2): 86 – 92

[14] Voas J M. Factors that affect software testability [C]//Proc of Pacific Northwest Software Quality Conference. Portland: IEEE, 1991: 235 – 247

[15] Yeh P L, Lin J C. Software testability measurements derived from data flow analysis [C]//Proc of the Second Euromicro Conference on Software Maintenance and Reengineering. Florence: IEEE, 1998: 96 – 102

[16] Nguyen T B, Delaunay M, Robach C. Testability analysis of data-flow software [J]. Electronic Notes in Theoretical Computer Science, 2005, 116(1): 213 – 225

[17] González Alberto, Pie éric, Gross H G. A model for the measurement of the runtime testability of component-based systems [C]//Proc of IEEE International Conference on Software Testing, Verification, and Validation Workshops. Denver: IEEE, 2009: 19 – 28

[18] Zhang Chengming, Zhang Jixian, Liang Yong, et al. Research on assessment method of testability for integrated information service platform [J]. Sensor Letters, 2010, 8(1): 191 – 197

[19] Salva Sébastien, Rabhi Issam. A preliminary study on BPEL process testability [C]//Proc of IEEE International Conference on Software Testing, Verification, and Validation Workshops. Paris: IEEE, 2010: 62 – 71

[20] Lamba C S. Testability models for structured programs [C]// Proc of 2nd International Conference on Communication Software and Networks. Singapore: IEEE, 2010: 420 – 425

[21] ISO/IEC 9126-1 Software engineering-product quality-part 1: quality model[S]

(编 辑:娄 嘉)