第33卷 第15期 计 算 机 工 程 2007年8月
Vol.33 No.15 Computer Engineering August 2007
·人工智能及识别技术· 文章编号：1000—3428(2007)15—0214—03 文献标识码：A 中图分类号：TP18

基于BDI的测试用例生成Agent模型

高 峰[1]，李晋惠[1]，王学通[2]
(1. 西安工业大学计算机科学与工程学院，西安 710032；2. 西安理工大学计算机科学与工程学院，西安 710048)

摘 要：在某些软件测试过程中，是由人工设计黑盒测试中的测试用例，这种方法人工重复劳动多、测试效率低。基于Agent理论中的BDI Agent的思想，给出了黑盒测试中自动生成测试用例的一个Agent模型，提出了一个使用该模型生成测试用例的算法。该模型能够依据黑盒测试中测试用例生成的基本原则和待测功能模块的功能说明，自动设计出相应的测试用例。使用该模型和算法生成了一个登录窗口的测试用例，解决了传统黑盒测试中的不足，具有重要的理论和实际意义。

关键词：BDI Agent；黑盒测试；测试用例

Agent Model of Design Test Case Based on BDI

GAO Feng[1], LI Jin-hui[1], WANG Xue-tong[2]
(1. Institute of Computer Science & Engineering, Xi'an Technological University, Xi' an 710032;
2. School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048)

【Abstract】Test-cases are currently designed manually in the process of software testing. There are much repetitive work and low test efficiency with this method. Therefore, this paper presents an agent model based on the ideal of BDI agent. This model can automatically design testc -ase in black _box testing based on the principals of designing test-case and the functions of module to be tested, it solves the traditional deficiency in black_ box testing.

【Key words】BDI Agent; black _box test; test case

在软件开发过程中，软件测试的工作量往往占软件开发工作总工作量的40%以上[1]，由此可见软件测试的重要性。软件测试方法有多种，其中黑盒测试是在不考虑代码结构的前提下，根据需求和运行环境对应用程序进行测试。目前在软件测试过程中，黑盒测试用例的设计基本上由测试人员手工完成，而大量复杂的测试用例设计完成后，如在测试中检测出错误，则提交的程序也要进行相应地修改，修改后可能还会引起新的错误，因此，还要由测试人员进行回归测试，即要检测已经发现的错误是否重现以及是否产生新的错误，如此反复，直到符合软件的质量要求。在此过程中，测试人员的重复劳动是很多的。

在Agent结构[2,3]的研究中，BDI Agent是一种实用推理结构，其推理过程和人们日常使用的方法类似，首先确定要达到的目标，然后确定如何达到这些目标。其基本成分是表示Agent信念(Belief)、意图(Intention)和愿望(Desire)的数据结构[4]。

据此，本文提出一个可以自动生成测试用例的基于BDI的Agent模型，该模型Agent可以根据设计测试用例的基本原则及程序的功能说明模拟测试人员设计测试用例的过程，并生成测试用例，以减少测试人员的重复劳动和提高测试的效率。

1 测试用例生成Agent模型描述

测试用例生成Agent（简称Agent _TC）模型<B，D, I>表示如下。其中，B∈Bel, D∈Des, I∈Int; Bel, Des, Int分别表示Agent _TC所有可能的信念、愿望、意图的集合。

(1)B=<B_p, B_F>

B在Agent _TC的BDI模型中应是一组Belief的集合，即Bel(Agent _TC, B)，其中，ψ表示谓词说明：

B_p：$\forall\psi\in B_P \Rightarrow$ Bel（Agent _TC, ψ），B_P是设计测试用例的各种基本原则的集合。

B_F：$\forall\psi\in B_F \Rightarrow$ Bel（Agent _TC, ψ），B_F是待测功能模块中的功能规格说明的集合。

(2)D=<D_{CI}, D_{CO}, D_{UI}, D_{UO}, D_O>

D在Agent _TC的BDI模型中应是一组Desire的集合，即Des(Agent _TC, D)，其中：

D_{CI}是测试用例中合理输入的集合。

D_{CO}是测试用例中合理输入对应的预期输出的集合。

D_{UI}是测试用例中不合理输入的集合。

D_{UO}是测试用例中不合理输入对应的预期输出的集合。

D_O是任何测试用例输入对应的不合理的输出的集合。

D_{CI}与D_{UI}允许包括一对或多个相互不相容的输入。

(3)I=< I_C, I_W>

I是经过执行已生成的测试用例后对程序运行结果正确与否的判断。执行测试用例输入部分（包括合理的输入和不合理的输入）后，若得到的输出结果与预期相应的输出一致，则程序运行结果正确，表示为I_C；否则运行结果不正确，表示为I_W，意味着程序需要修改。虽然I_C和I_W表示符号不同，但是它们都是对于程序运行结果的判断，避免了意图冲突的可能性。

约定上述B、D、I满足典型的BDI约束关系[5]。

作者简介：高 峰(1976－)，女，硕士，主研方向：人工智能，软件测试；李晋惠，教授；王学通，讲师

收稿日期：2006-08-20 **E-mail：**tjgfgf@126.com

2 算法描述

该模型生成测试用例的算法可分为以下几个步骤。

Step1

初始化操作。将模型中B、I置初始状态，形成Agent _TC初始的关于当前环境的信息和意图。B_F取决于程序的功能规格说明，B_p取决于黑盒测试的设计测试用例的基本原则，I则为对程序运行结果是否正确的判断。

首先程序的规格说明要细化到每个功能模块的不可再分的原子功能点，Agent _TC 要掌握程序的原子功能点作为自己信念的组成部分。

其次由于软件工程的所有过程中除测试外都是“建设性”的，而软件测试则是“破坏性”的，因此测试用例的设计要包含输入和输出两部分，其中测试用例的输入要包含合理的输入和不合理的输入两部分，而不合理的输入则体现了测试的“破坏性”。黑盒测试的测试用例的设计基本原则有等价类划分法、边界值分析法、错误推测法等。

(1)等价类划分法

把全部输入数据合理地划分为若干个等价类，在每一个等价类中取一个数据作为测试的输入条件，即用少量的代表性测试数据，取得较好的测试效果。

1)如果输入条件规定了取值范围$[a, b]$，或值的个数 n，则可以确立一个有效等价类和两个无效等价类：

一个有效等价类是：$\forall x \in [a,b]$，或者$\forall x=n$；

两个无效等价类是：$\forall x \notin [a,b]$，即$\forall x<a$，$\forall x>b$；或者$\forall x \neq n$ 即$\forall x<n$，$\forall x>n$。

2)如果输入条件规定了输入值的集合 A = {a,b,c,….}，或者是规定了“必须如何”的条件，这时可以确立一个有效等价类和一个无效等价类：

一个有效等价类是：$\forall x \in A$ 或者$\forall x$ 满足所规定的条件；

一个无效等价类是：$\forall x \notin A$ 或者 $\forall x$ 不满足所规定的条件。

3)如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类：

一个有效等价类是：$\forall x = F$或T ；

一个无效等价类是：$\forall x \neq F$或T 。

4)如果规定了输入数据的一组值$\{A,B,C\}$，而且程序要对每个输入值分别进行处理，这时可以为每一个输入值确立一个有效等价类，并针对这组值确立一个无效等价类：

每个输入值的一个有效等价类：$\forall x \in A$，$\forall y \in B$，$\forall z \in C$；

总体的一个无效等价类：$\forall x \notin A$，$\forall y \notin B$，$\forall z \notin C$。

(2)边界值分析法

是对等价类划分法的补充。针对各种边界情况设计测试用例，应当选取正好等于，刚刚大于，或刚刚小于边界的值作为测试用例，而不是选取等价类中的典型值或任意值作为测试数据，可以查出更多的错误。

1)如果输入条件规定了值的个数$[m, n]$，则用最大个数 n、最小个数 m、比最大个数多 1(n+1)、比最小个数少 1(m-1)的数作为测试数据。

2)如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

3)如果程序的规格说明给出的输入域或输出域是有序集合（如有序表，顺序文件等），则应选取集合的第 1 个元素和最后 1 个元素作为测试用例。

(3)错误推测法

依靠经验和直觉推测程序中可能存在的各种错误，从而有针对地编写检查这些错误的例子：

如输入数据为 0 或输出数据为 0 时容易发生错误的情形，因此可以选择输入数据为 0，或输出数据为 0 的例子作为测试用例。

对于 Agent _TC 来说，它应该具有推测错误的能力，即 Agent _TC 应该能够根据测试经验及推理来推测程序可能有的错误。

总之设计测试用例的基本原则是一定的，可按照软件复用的原则可以将B_p封装在一个类中，以便重复使用。

Step2

获得外界信息，根据 Agent _TC 当前的信念和感知输入，修正已有的信念并确定新的信念的集合。

当进行黑盒测试之前，Agent _TC 还要获得最新的功能规格说明，通过信念修正函数 brf，来修正自己当前信念中的功能说明部分。

信念修正函数 brf：Ψ(Bel)×P→Ψ(Bel)，其中，P 表示感知。

Step3

选择生成Agent _TC可能的愿望。根据Step1 中的初始化操作和Step2 中的修正信念以及当前的意图，通过选择生成函数options形成Agent _TC模型中的D_{CI}、D_{CO}、D_{UI}、D_{UO}、D_O。

选择生成函数 options：Ψ(Bel)×Ψ(Int)→Ψ(Des)。

Step4

Agent _TC根据选择生成的D_{CI}、D_{CO}、D_{UI}、D_{UO}、D_O，得出程序正确运行与否的结果I，以作为完整的测试用例。

当应用 Agent _TC 设计出的测试用例检测出错误后，程序开发人员需要修改出错模块，修改完成后，还要进行回归测试，此时若无功能的增减，则可继续使用 Agent _TC 模型设计的测试用例重新进行测试，若程序有功能上的增减，则需要 Agent _TC 回到 Step2 中，重新修正新增减的原子功能规格说明，然后再循环执行 Step3~Step4，再生成相应的测试用例。而且 Agent _TC 经过大量的测试得到测试经验后，有必要自己能够回到 Step1 中充实自己的错误推测能力。

3 模型的应用

在某些网站和 MIS 系统中，通常有用户登录的简单功能，下面将 Agent _TC 模型应用到上述功能模块中生成测试用例以作简单说明。

用户登录模块的功能规格说明是：该模块中应该具有填写用户名(user _name)、用户密码(password)、校验码(check _code)功能。其中B_P为按算法Step1 中描述建立，B_F为：

用户名：要求由字母和数字组成，只能以字母开头，不能为空；

用户密码：要求由字母和数字组成，长度 6 位，且不能为空；

验证码：要求用户输入的字符与显示在校验图片中的字符相同。

(1)Agent _TC 模型在此模块中按照 Step1 初始化 B，其中对于相应功能分解出的原子功能说明如表 1 所示。

表 1　原子功能说明

用户名 user _name	user _name 字符类型 type ∈{字母，数字}
	user _name 首字符 initial ∈{字母}
	user _name 不能为空
用户密码 password	password 字符类型 type ∈{字母，数字}
	password 长度 length = 6
验证码 check _code	check _code 与显示在校验图片中的字符相同(图 1)

图 1　验证码示例

(2)Agent _TC 进入到 Step2 根据外界感知的信息和已有的信念进行信念修正。假设在初始化 Agent_TC 的信念后，有新的功能说明如：password 长度 length 改为 8 位，则 Agent_TC 需要根据信念修正函数修正原有的信念为 password 长度 length = 8。如果还有其它功能说明增减，则 Agent_TC 仍需要进行信念修正操作。

(3)Agent_TC进入Step 3，这里Agent_TC的最终意图是为了检测程序运行所给功能的正确与否，因此，Agent_ TC 根据上面 2 个步骤中的操作结果及当前意图选择生成其愿望。如检测输入user _name类型时，为了检测程序运行正确，Agent_TC需要根据选择生成函数options得出相应的愿望，即Ψ(Des)＝{ D_{CI}, D_{CO}, D_{UI}, D_{UO} }，其中

D_{CI} ＝{user _name　type ∈{字母，数字}}，

D_{CO} ={提交成功}；

D_{UI} ={user _name　type ∉{字母，数字}}，

D_{UO} ={程序报错}。

而为了检测程序运行不正确，则其愿望为

Ψ(Des)＝{ D_{CI},D_{UI},D_O }，其中

D_O={ D_O^1, D_O^2}，

D_{CI} ＝{user _name　type ∈{字母，数字}}，

D_O^1 ={提交不成功}；

D_{UI} ={user _name　type ∉{字母，数字}}，

D_O^2 ={程序接收了不合法的输入，未报错}。

其它功能点的用例生成与上述类似。

(4)Agent_TC 进入 Step4，得出完整的测试用例：

1) D_{CI}：user _name　type ∈{字母，数字}；
　D_{CO} ：提交成功；I_C ：程序运行正确。

2) D_{CI}：user _name　type ∈{字母，数字}；
　D_O ：提交不成功；I_W ：程序运行不正确。

3) D_{UI}：user _name　type ∉{字母，数字}；
　D_{UO} ：程序报错；I_C ：程序运行正确。

4) D_{UI}：user _name　type ∉{字母，数字}；
　D_O ：程序接收了不合法的输入，未报错；
　I_W ：程序运行不正确。

5) D_{CI} ：user _name　initial ∈{字母} & user _name　type ∈{字母，数字}；
　D_{CO} ：提交成功；I_C ：程序运行正确。

6)D_{UI}：user _name　initial ∈{字母} & user _name type ∉{字母，数字}；
　D_{UO}：提交不成功；I_C ：程序运行正确。

7) D_{UI}：user _name　initial ∉{字母}；
　D_{UO}：提交不成功；I_C ：程序运行正确。

8) D_{UI}：user _name　initial ∉{字母}；
　D_O：提交成功；I_W：程序运行不正确。

9) D_{UI}：user _name为空；
　D_{UO}：提交不成功；I_C ：程序运行正确。

10)D_{UI}：user _name为空；
　D_O：提交成功；I_W：程序运行不正确。

11)D_{CI}：password　length = 6 位；
　D_{CO}：提交成功；I_C ：程序运行正确。

12)D_{CI}：password　length = 6 位；
　D_O：提交不成功；I_W ：程序运行不正确。

13)D_{UI} ：password　length ≠6 位；
　D_{UO}：提交不成功；I_C ：程序运行正确。

14)D_{UI}：password　length ≠6 位；
　D_O：提交成功；I_W ：程序运行不正确。

15)D_{UI}：password　length =0 位；
　D_O：提交成功；I_W ：程序运行不正确。

16)D_{UI}：password　length =0 位；
　D_{UO}：提交不成功；I_C ：程序运行正确。

17)D_{CI}：check _code输入与验证码相同，
　D_{CO}：提交成功；I_C ：程序运行正确。

18)D_{CI}：check _code输入与验证码相同；
　D_O：提交不成功；I_W ：程序运行不正确。

19)D_{UI}：check _code输入与验证码不同；
　D_{UO}：提交不成功；I_C ：程序运行正确。

20)D_{UI}：check _code输入与验证码不同；
　D_O：提交成功；I_W ：程序运行不正确。

21)对于 password 字符类型 type ∈{字母，数字}的检测用例生成与 user _name 字符类型 type ∈{字母，数字}的用例设计类似，在此不再赘述。

由于用户登录模块在许多的 C/S 或 B/S 系统中很常见，因此只需要对上述的 Agent _TC 设计的测试用例中的一些接口，如字符类型、长度等做简单修改即可以重复使用。

4　结束语

本文提出的测试用例生成 Agent 模型及算法设计不仅将 BDI 用于模型的表示，而且还使用了 BDI 之间的约束关系。通过文中所举的例子充分说明，该模型对黑盒测试中的测试用例的生成能够起到一定的辅助作用，并简化测试人员的重复劳动，尤其在反复地回归测试过程中，提高了测试效率。

参考文献

1 张海藩. 软件工程导论[M]. 北京: 清华大学出版社, 1997.

2 Weiss G. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence[M]. MIT Press, 1999.

3 史忠植. 智能主体及其应用[M]. 北京: 科学出版社, 1998.

4 王文杰. 人工智能原理与应用[M]. 北京: 人民邮电出版社, 2004.

5 Haddadi A S. Communication and Cooperation in Agent Systems[M]. Berlin: Spring-Verlag, 1996.