第 35 卷 第 7 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.7 Computer Engineering April 2009
·软件技术与数据库· 文章编号：1000—3428(2009)07—0055—04 文献标识码：A 中图分类号：TP311

一种改进的 GUI 测试框架 DART

辛敏杰，高建华
(上海师范大学计算机科学与技术系，上海 200234)

摘 要：针对当前的图形用户界面自动化测试中存在的局限性，提出一种每日自动回归测试(DART)框架的改进方案。研究 DART 流程、事件流图，包含了事件关联图的方法，实现对事件流图的优化，并采用权重选择法对 DART 方法进行改进，使之成为一种高效、简洁的自动冒烟测试框架。

关键词：图形用户界面；图形用户界面测试；每日自动回归测试

Improved Framework DART for GUI Test

XIN Min-jie, GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234)

【Abstract】 In response to the limitation in Graphic User Interface(GUI) automated test, this paper proposes an improved framework for Daily Automated Regression Tester(DART). DART process and event-flow graph is studied. And event-interaction graph which can simplify the event-flow graph is also included. In addition, DART is improved as an efficient and brief automated smoking test framework by weight selection.

【Key words】 Graphic User Interface(GUI); GUI test; Daily Automated Regression Test(DART)

1 概述

图形用户界面(Graphic User Interface, GUI)被广泛地用于应用软件中，但其大量使用也为软件的开发和测试带来了极大的挑战。在今天的软件开发中，GUI 可占到全部代码 60%以上。因此，GUI 测试(GUI test)在软件开发和测试中具有重要的地位。

由于 GUI 软件经常采用团队异地开发，因此对不同编程者所编制的程序之间的整合与测试造成了不便。而且，在软件质量中，最注重的是软件正确性(correctness)、可靠性(reliability)和效率(efficiency)。一个基于快速反馈的质量保证机制(rapid-feedback-based quality assurance mechanisms)将会有效地解决这样的情况。

当前，GUI 测试的理论研究日趋完善。文献[1]提出的每日自动回归测试(Daily Automated Regression Test, DART)中的冒烟测试(smoke test)框架为以上情况提供了一个有效的解决方案。该方案提出了 DART 测试框架，并详细研究了在 DART 框架下不同测试条件对 GUI 测试产生的影响。其主要研究对象包括：

(1)冒烟测试集(smoke test suite)的大小对测试的影响。

(2)预测的复杂度对测试的影响。

(3)DART 方法中可检测出及不可检出的错误的特征。

文献[2]认为，软件测试是一项劳动密集型和资源密集型的工作，测试将会占到开发成本的 50%~60%。当前 GUI 测试中的主要困难有：

(1)覆盖标准(coverage criteria)：传统的基于代码的覆盖标准无法再使用到 GUI 用户事件和应用中，而过于全面的覆盖程度会导致测试效率降低。

(2)测试预测(test oracle)：无法一次性进行测试预测，在 GUI 测试中预测必须与实际测试交叉进行。

(3)回归测试(regression test)：如果程序修改后 GUI 布局发生变化，将会导致原来的测试用例无效。

(4)GUI 测试过程(GUI test process)：传统的测试过程无法适应 GUI 测试。

本文在分析 DART 中的局限性的基础上，针对 DART 测试随机性错误能力不足的局限，提出结合权重测试的方法加大测试用例中权重较大事件的测试频率，从而达到改进测试效果的目的。

2 每日自动回归测试

冒烟测试[3](smoke test)指软件(或软件的一个部分)在编制后或一次关键改动后的首次运行。

DART 是一种已被实现的软件测试框架。开发者在白天编制、修改程序代码，晚上 DART 自动运行 GUI 测试应用(GUI Application Under Test, AUT)，创建出 GUI 冒烟测试用例。

2.1 DART过程

测试者使用 DART 过程实现自动化测试。DART 过程概括如下：

(1)开发者(或测试者)选择 AUT。开发者需要选择编译 AUT 所需的源文件和库模块。DART 为这些 AUT 元素建立内部标识。这个版本的 AUT 成为“基准 AUT(baseline AUT)”。

(2)DART 分析(基准)AUT 中的 GUI 结构(使用 GUI 提取器模块)，自动遍历所有窗体中的 GUI，并提取 GUI 对象(称为工具集)和对象的属性。内部 AUT 表示为一个三元组(对象，属性，属性值)以及事件流图(event-interaction graphs)。

(3)DART 计算出可能在 AUT 上执行的冒烟测试用例总数。由测试者决定执行测试用例的数量(测试者并不决定测试

基金项目：国家自然科学基金资助项目(60673067)
作者简介：辛敏杰(1983－)，男，硕士研究生，主研方向：软件可靠性设计；高建华，教授、博士
收稿日期：2008-09-10 **E-mail**：norman.xin@gmail.com

何种测试用例，仅决定数量，这使得测试快而有效)。默认状态下，执行长度为 1 和 2 的测试用例。

(4)DART 用测试用例生成器(test case generator)自动生成测试用例。

(5)预测生成器(test oracle generator)自动为每个用例创建期望输出，用于检验下一版本 AUT 的正确性。到此，一个冒烟测试集(smoke test suite)生成。

(6)开发组根据修改请求(change requests)和错误报告(bug reports)修改 AUT。

(7)操作系统任务管理器启动 DART，从而启动 AUT。DART 自动使用代码执行器(code instrumenter)执行 AUT 源代码。

(8)测试用例通过修改后的 AUT 被测试用例执行器(test case executor)执行，将输出结果与储存的期望结果(第(5)步)进行比较，生成执行报告(包括成功或者不成功)。覆盖报告也被生成，报告中说明了哪些语句和分支被执行。这些结果通过电子邮件发送给开发者。生成一个基于 Web 的公告板。公告板上提供用例测试结果总结以及测试结果的细节。

(9)次日，开发者检验报告及不成功的测试用例。

(10)开发者递交错误报告。

冒烟测试过程在步骤(3)~步骤(10)之间循环，如有必要，代码冒烟测试也会加入此循环。DART 过程如图 1 所示。

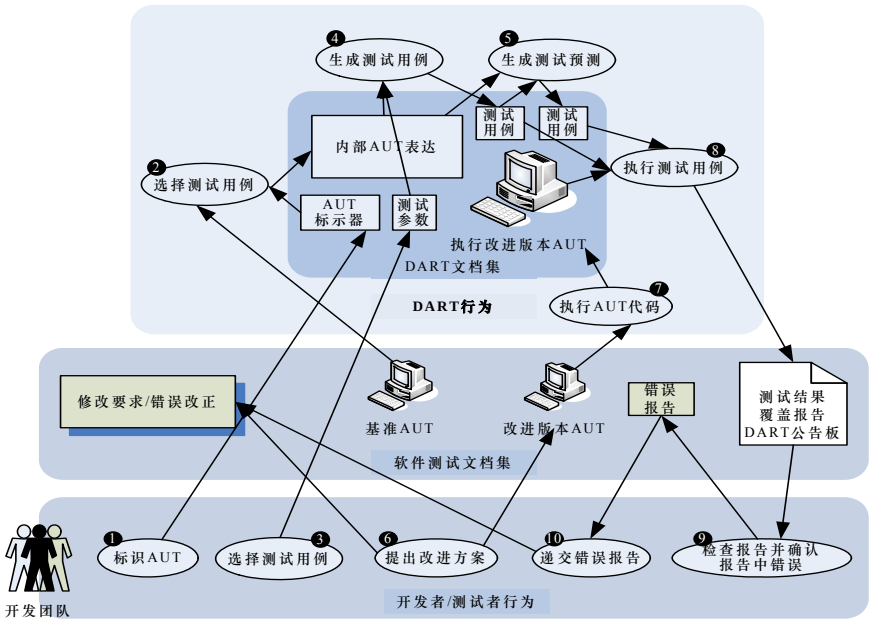

图 1　DART 过程

2.2　DART过程中测试用例的选择

DART 中确定测试用例的基本方法为改进后的事件流图。

2.2.1　事件流图(EFG)

GUI 是由许多组件构成的，其中的每一个事件都和 GUI 中某一个组件相联系，这些事件之间的联系可用事件流图的形式描述。事件流图中包括 GUI 中对应的所有事件以及这些事件之间的相互关系，可以将其描述为一个二元组<*N*, *E*>。*N* 是节点集合，代表 GUI 中所有的事件，每个节点表示 GUI 中的一个事件。*E* 表示直接连接节点的边，边(n_x, n_y)表示 n_y 代表的事件可以接着 n_x 代表的事件进行。例如，微软计算器软件中主窗口和 Replace 窗口的 EFG 表示如图 2 所示。

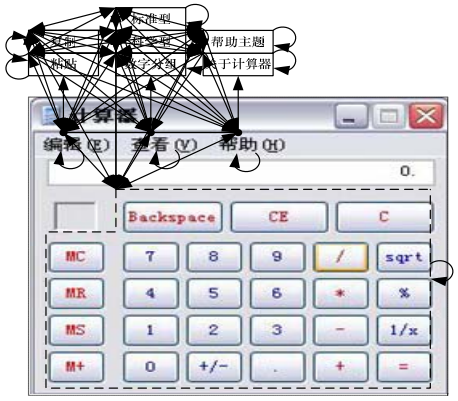

图 2　Windows 计算器 EFG 例子

2.2.2　事件关联图(EIG)

事件流图是 GUI 测试中一种重要的测试方法，理论上通过遍历事件流图，可以达到测试路径的完全覆盖。但是，它也是有局限性的，例如事件流图本身往往很复杂，很难应用于实际测试中，因此，必须对它进行相应的改进，使它更加贴近实际应用，更符合 GUI 测试的需要。

定义　能够影响非结构性事件(nonstructural events)，即不会造成 GUI 结构影响的事件或软件中关闭的窗口的事件称为系统关联事件(system-interaction events)。

A. M. Memon 指出，软件中绝大部分事件为系统关联事件。例如，微软 Word 中包括 4 210 个事件，其中，有 3 588 个是系统关联事件。而且，大多数非系统关联事件代码用于 GUI 结构操作，大多数代码自动生成，标准性较高，错误出现概率较小。那么，简化 EFG 的有效途径就是将系统关联事件加入测试用例中。以下为从事件流图到事件关联图的转换算法。

```
N   /* Nodes set of EIG */
E   /* Edges set of EIG */
PROCEDURE :: GenerateEIG(
   Event Flow Graph (N, E)) {
     N = N
     E = E
FORALL n ∈ N DO
   start(n) = { n_i | (n, n_i) ∈ E, and n ≠ n_i }
   end(n) = { n_i | (n_i, n) ∈ E, and n ≠ n_i }
FORALL n ∈ N DO
   IF EventType(n) ≠ system- interaction
FORALL n_x ∈ end(n) DO
      FORALL n_y ∈ start(n) DO
         E = E ∪ (n_x, n_y)
         IF n_x ≠ n_y
            start(n_x) = start(n_x) ∪ {n_y}
            end(n_y) = end(n_y) ∪ {n_x}
FORALL n_x ∈ end(n) DO
      remove n from start(n_x)
FORALL n_y ∈ start(n) DO
      remove n from end(n_y)
remove n from N
remove all edges (n, n_i) from E
remove all edges (n_i, n) from E}
```

图 3 为从图 2 转换的 EFG。可以看到，在“帮助”菜单以及“查看”菜单中的“标准型”和“科学型”可造成 GUI 结构改变而被去除，从而大大缩小了测试用例的选取范围。长度为 1 的事件序列即枚举所有节点。长度为 2 的事件序列即枚举所有节点及其相邻节点。图中，系统关联事件={计算器面板按钮，复制，粘贴，数字分组}。另外，由于事件“复制”、“粘贴”必须经过非系统关联事件菜单栏事件“编辑”，因此“编辑”也被选取。系统关联事件={计算器面板按钮，复制，粘贴，数字分组}。由于事件“复制”，“粘贴”必须经过非系统关联事件菜单栏事件“编辑”，因此“编辑”也被选取。

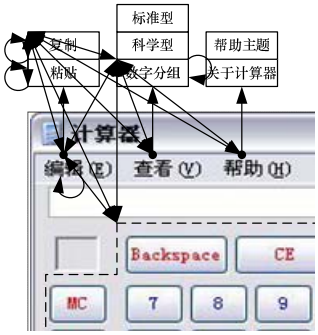

图 3　图 2 事件流图到事件关联图的转换

3　DART过程的局限性及其改进

DART 过程是一种高效的软件测试框架，它通过优化测试用例数量，使冒烟测试的速度加快。但是，DART 过程的实践仍停留在实验阶段，尚未投入商业应用。而且，DART 过程在检错有效性上也有一定的局限性。

3.1　DART过程的局限性

DART 测试过程的主要局限性如下：

(1)笔者的研究案例中仅仅采用了 Java 应用，而对 Win32 的应用测试结果可能不同。

(2)对于一些强调业务逻辑(business logic)的应用，DART 没有考虑将 GUI 在其业务逻辑中进行测试。

(3)在 DART 测试中各组件测试的权重及次数相同，这可能使一些随机发生的错误检测的有效性下降。

3.2　DART过程的改进

第(2)项和第(3)项局限性由测试用例的覆盖范围及长度局限所致。解决这个问题就是解决测试用例的选取问题。在 DART 过程中，同一窗口中的 GUI 控件权重相同，因此，在测试用例中，不同控件被执行的次数也相同。在软件设计中，一些错误会随机发生，通过有限次的测试可能仍然无法测出该随机错误。一般，GUI 对象执行得越多，随机类型错误越有可能出现。同样，一个 GUI 如果增加测试次数，那么随机错误检出的可能性就越大。对于一些重要 GUI 对象或一些使用频率高的研究对象，有必要增加其测试次数。

GUI 测试中随机测试(random test)[4]的方法为解决这个问题提供了一个可能的方法。通常在软件测试中无法进行穷举测试，而只能采取随机测试的方法，即在尽量不牺牲测试效率的前提下，有选择地测试对象。在 GUI 随机测试中，须通过一定的方法选择测试对象。

文献[4]描述了一种选择测试对象的方法——权重选择法。在选择测试对象时，根据对象的复杂性，例如，对于一个函数，计算复杂度越高或分支结构越多，权重越大，则被选为测试用例的可能性越大。

此方法可以运用到 DART 中，对于不同的 GUI 对象，根据其重要性(如对象使用频率、代码复杂性或在业务逻辑的重要性)赋予不同的权重，在生成测试用例时，根据权重决定包含该对象测试用例的数量，从而提高该对象测试次数，达到提高随机错误检出率的目的。

根据以上所述，将测试分为 2 级：

L1：普通 GUI 测试，即根据事件关联图产生测试用例并测试；

L2：权重 GUI 测试，在 L1 完成的基础上根据权重产生测试用例并测试。

对于不同的软件特点，选取不同的测试级别，一般软件执行 L1~L2 测试。由此，在 DART 过程中的改进如下：

(1)DART 分析(基准)AUT 中的 GUI 结构(使用 GUI 提取器模块)，自动遍历所有窗体中的 GUI，并提取 GUI 对象(称为工具集)和对象的属性。内部 AUT 表示为一个四元组(对象，属性，属性值，权重)以及事件流图。

(2)DART 计算出可能在 AUT 上执行的冒烟测试各级测试用例总数(包括权重测试)。由测试者决定执行测试用例的数量(测试者并不决定测试何种测试用例，仅决定数量，使测试快而有效)。在 L1 测试下，执行长度为 1 和 2 的测试用例。在 L2 测试中，根据权重的不同，测试用例长度发生变化。

3.3　研究案例：仿Windows 计算器L2 测试用例

以仿 Windows 自带计算器为例，在此根据事件代码复杂程度决定事件权重大小(代码用 C#模仿 Windows 计算机界面及功能编写，界面与 Windows 计算器相同，但 backspace, % 及存储功能未实现)。

GUI 事件的权重值选取是一个很困难的过程，可以结合测试人员及领域专家的经验、以往的数据以及测试过程所获得的经验数据，通过对以下几个因素进行综合考虑估计事件的权重值。因素之间相互关联，对各因素获取的有关信息越详细，所估计的 GUI 事件权重值越可靠。考虑的因素主要有如下 4 种[5]：

(1)功能模块在 GUI 软件系统中的重要性、安全性；

(2)功能模块的用户使用频度；

(3)功能模块的复杂度；

(4)测试过程中功能模块的出错概率。

一般，重要模块所包含的 GUI 事件的权重值大，用户使用频度高的功能模块中 GUI 事件的权重值大，复杂度高的功能模块中 GUI 事件的权重值大，以前的测试中出现过错误的功能模块中 GUI 事件的权重值大。

仿 Windows 计算器控件分支数、语句数及权重结果如表 1 和图 4 所示。为表述方便，约定权重取值范围为 0~1。

表 1　仿 Windows 计算器控件分支数、语句数及权重

GUI 控件	分支数	语句数	权重
数字按钮 0-9	12	13	1.33
=	7	30	1.00
/	3	14	0.44
*	3	13	0.43
sqrt	3	11	0.41
-	2	16	0.36
+	2	12	0.32
1/x	2	11	0.31
+/-	1	8	0.18
.	1	6	0.16
‘复制’	1	2	0.12
CE，C	0	8	0.08
‘粘贴’	0	1	0.01

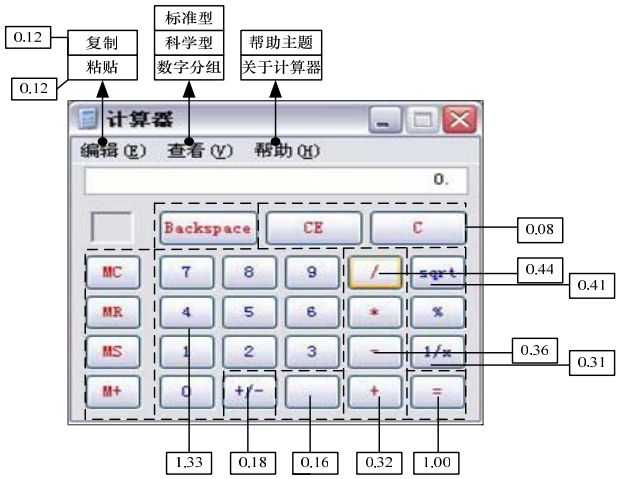

图 4　计算器各控件权重

权重取定后，用户可决定测试用例长度及数量。其中，代码越复杂，权重越大，事件被包含进用例的概率越大，测试次数就越多。一个事件 i 在用例中的出现次数 n_i 可表达为

$$n_i = N \times (w_i / \sum w)$$

其中，N 为用户所定测试用例中时间的个数；w_i 为事件 i 的权重大小；w 为所有事件权重总和。以仿 Windows 计算器程序为例，权重及 p_i 如表 2 所示。

表 2　仿 Windows 计算器控件权重及 p_i

GUI 控件	权重	$p_i=w_i/\sum w$
数字按钮 0～9	1.33	0.258
=	1.00	0.194
/	0.44	0.085
*	0.43	0.083
sqrt	0.41	0.080
-	0.36	0.070
+	0.32	0.062
1/x	0.31	0.060
+/-	0.18	0.035
.	0.16	0.031
‘复制’	0.12	0.023
CE，C	0.08	0.015
‘粘贴’	0.01	0.002

按照表 2，用户设定用例中包含事件数为 100，用例长度为 2，那么事件“数字按钮 0~9”在用例中为 100×0.258≈26 个，事件“=”在用例中为 100×0.194≈19 个，依此类推。因此，在所有这些测试用例中，包含事件“数字按钮 0~9”的集合可为{(数字按钮 0~9，数字按钮 0~9)，(数字按钮 0~9, =)，(数字按钮 0~9, /)，(数字按钮 0~9, *)(数字按钮 0~9，sqrt)，(数字按钮 0~9, -)(数字按钮 0-9, +)，(数字按钮 0~9, 1/x)，(数字按钮 0~9, +/-)，(数字按钮 0~9, .)，(数字按钮 0-9, ‘复制’)，(数字按钮 0~9, CE/C)，(数字按钮 0~9, ‘粘贴’)，(=，数字按钮 0~9)，(/，数字按钮 0~9)，(*，数字按钮 0~9)，(sqrt，数字按钮 0~9)，(-，数字按钮 0~9)，(+，数字按钮 0~9)，(1/x，数字按钮 0~9)，(+/-，数字按钮 0~9)，(.，数字按钮 0~9)，(‘复制’，数字按钮 0~9)，(CE/C，数字按钮 0~9)}。

4　结束语

DART 框架是一种高效、简洁的自动化冒烟测试方法，其中运用事件流图的改进——事件关联图优化了测试用例的选择，大大加快了冒烟测试的执行时间。但是，DART 仍有其局限性，包括随机错误的检测及业务流程的体现。随机测试中的权重选择用例为解决这一问题提供了一种可能的方法，不同等级的测试适用于不同特征的软件。但是，由于条件所限，改进后 DART 的效率和有效性仍有待进一步研究。

参考文献

[1] Memon A M, Qing Xie. Studying the Fault-detection Effectiveness of GUI Test Cases for Rapidly Evolving Software[J]. IEEE Transactions on Software Engineering, 2005, 31(10): 884-895.

[2] Memon A M. GUI Testing: Pitfalls and Process[J]. IEEE Computer, 2002, 35(8): 87-88.

[3] McConnell S. Daily Build and Smoke Test[J]. IEEE Software, 1996, 3(4): 143-144.

[4] Daboczi T, Kollar I, Simon G, et al. How to Test Graphical User Interfaces[J]. IEEE Instrumentation & Measurement Magazine, 2003, 6(3): 27-33.

[5] 胡积平，李军义. 一种改进的自动规划 GUI 测试用例生成方法[J]. 科学技术与工程, 2006, 6(22): 3575-3580.

编辑　张正兴

（上接第 54 页）

[3] Buyya R, Abramson D, Venugopal S. The Grid Economy[J]. Proceedings of the IEEE, 2005, 93(3): 698-714.

[4] Li Yuanhui, Zhao Depeng, Li Jun. Scheduling Algorithm Based on Integrated Utility of Multiple QoS Attributes on Service Grid[C]// Proc. of the 6th International Conference on Grid and Cooperative Computing. Washington D. C., USA: IEEE Computer Society, 2007: 288-295.

[5] 罗　红，慕德俊，邓智群，等. 网格计算中任务调度研究综述[J]. 计算机应用研究, 2005, 22(5): 16-19.

[6] Wolski R, Spring N T, Hayes J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing[J]. Journal of Future Generation Computing System, 1999, 15(5/6): 757-768.

[7] Gong Linguo, Sun Xianke, Watson E F. Performance Modeling and Prediction of Nondedicated Network Computing[J]. IEEE Trans. on Computers, 2002, 51(9): 1041-1055.

[8] Ali S, Siegel H J, Maheswaran M, et al. Task Execution Time Modeling for Heterogeneous Computing[C]//Proc. of the 9th Heterogeneous Computing Workshop. Washington D. C., USA: [s. n.], 2000: 185-199.

编辑　顾逸斐