移动应用测试-GUI 测试综述

摘要

GUI 是计算机软件与用户进行交互的主要方式。当前针对移动应用测试研究的一个关键方向就是移动应用的 GUI 测试。近年来，移动应用 GUI 测试的问题吸引了大量的研究者的目光。本文将简要归纳 GUI 测试方面研究中的相关问题并重点归纳移动应用 GUI 测试方向研究的关键问题，对一般软件 GUI 测试简单介绍，对移动应用 GUI 测试研究的主要进展进行重点介绍和总结，并将二者比较，企图得到移动应用 GUI 测试的更好解决方案与方法。

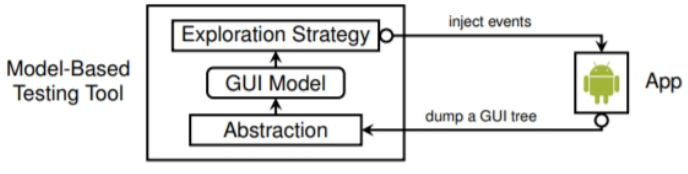
关键词 移动应用 GUI 测试 自动化测试

1 引言

当今时代，GUI 软件和移动设备与人们紧密相连。自 1981 年 Xerox 公司推出首款图形用户界面电脑 Xerox 8010 以来，通过 GUI 操纵软件已经成为人们使用软件的主要方式。除了普通应用，GUI 软件也广泛应用于银行医疗航空等安全可靠关键（security-safe-critical）领域。GUI 给人们带来了更加便捷的使用体验，但是却给软件开发带来了更大的工作量和复杂性。据统计，GUI 软件用于 GUI 本身的代码量平均占到软件总代码量的 48%，有时更高达总代码量的 60%。因此，对 GUI 软件的测试成为了极具挑战性的任务。对于当前的移动设备，GUI 软件覆盖率更是接近百分之百。

随着移动设备普及率迅速增长，移动应用的数量和规模逐年膨胀，对移动应用测试的需求也大规模增长。当下，移动应用无处不在，人们每天都在与移动应用打交道，且现在移动设备对用户友好度追求更加极致，手机等移动设备为几乎全部采用图形界面，而 GUI 正是与用户进行交互的主要方式，直接影响用户体验。移动应用 GUI 的测试由于其凸现出来的重要性，已日渐引起学术界和工业界的兴趣和重视。然而，目前关于移动应用 GUI 测试的研究还处于初级阶段：很多问题还没有解决，GUI 软件测试依然需要较高人工成本，目前的技术还不能满足保证软件质量的实际需求。因此，如何提高移动应用 GUI 测试效率和准确性测试的可靠性和正确性也成为了一个重要问题。

当前，对于保证移动应用 GUI 测试的可靠性和正确性，已经有了不少探索。相关的论文工作也层出不穷，本文将针对近年来的移动应用 GUI 测试进行系统的介绍和总结。



【图 1】经典的基于模型的 Android 应用 GUI 测试工作流

2 GUI 测试的定义

GUI 测试，是图形用户界面的缩写，或用户可见的应用程序的一部分。GUI 可能包括诸如菜单、按钮、文本框、图像等元素。当今的 GUI 测试不在局限在计算机的桌面，他可能是在所有主要的平台上可用的移动应用程序。测试人员必须通过跨平台测试来识别缺陷并确保应用程序满足所有要求。因此，GUI 测试是指测试用户可见的应用程序的功能。GUI 测试还会确认外观元素（如字体和图像）是否符合设计规范和要求。GUI 测试发生在系统测试的级别。

3 提升移动应用 GUI 测试可靠性和正确性的解决方案

最简单的方法是模糊测试，它产生随机事件来探索自动测试的行为并检测故障。然而，模糊测试的一个主要局限性是，由于随机性的本质，会产生大量的冗余事件。此外，跟踪和调试可疑路径到检测到的故障比系统方法更困难。因此我们企图寻求其他方面的解决思路或可以改善模糊测试的方法。

3.1.1 基于反馈的探测策略【1】

这是一种有效的基于模型的 Android 应用程序 GUI 测试技术。为了避免局部和重复的探索,该团队的方法将等效部件打包成组，并且设计一种新颖的，基于反馈的勘探策略,这将通过基于那些已经出发的动作的执行结果，动态调整动作的优先级,并倾向于选择那些可以到达新状态的应用程序的行为。该团队研发了 CrawlDroid 的工具实现了该技术，并进行了实验。

Table 1: UI Action and Initial Priority

Action type	Init. score	Action type	init. score
click	50	longClick	50
scroll	40	inputText	-1
swipe	10	pressBack	10
rotate	10		

基于模型的测试将根据预定义的遍历策略(例如深度优先搜索、广度优先搜索)对每个确定可执行小部件执行操作。该方法设计了一种新的基于反馈的探索策略，它将在一种状态下的小部件进行分组，并分配一个优先级值给每一个该组支持的动作。根据一个动作的执行结果，探索策略会调整动作的优先级(属于一个组)。通过动态地分组小部件，调整动作的优先级。

Algorithm 1: Overall Exploration Algorithm

```
1  begin
2      startApp(); SFG ← SFG_init;
3      S ← TransToHighPriorityState(SFG);
4      while constraintSatisfied(CC) do
5          gr ← biasSelectGroup(S);
6          ac ← randSelAction(gr); N ← triggerAction(S, ac);
7          ac.select ← 1 for app action;
8          compare(N, S);
9          if state does not change then
10             decGroupPriority(S, gr);
11             if N.priority < threshold then
12                 S ← transitState(SFG, N);
13         else
14             SFG.update(N, ac);
15             if N is a new state then
16                 incGroupPriority(S, gr);
17                 computeActionGroup(N); S ← N;
18 Procedure biasSelectGroup(S)
19     sumPriority ← 0; arrayPriority ← []; groups ← [] ;
20     computeGroupAction(S); groups ← getActionGroups(S);
21     foreach gp ← groups do
22         sumPriority ← sumPriority + gp.priority;
23         arrayPriority.push(sumPriority);
24     random ← Random.nextInt(sumPriority);
25     group ← maping(arrayPriority, random, groups);
26     return group;
```

3.1.2 启发

现有的基于模型的测试技术的一个主要限制是，它们花费大量的时间来探索 AUTs 的一些局部，而没有获得覆盖改进。这种方法可以避免局部探测，并有更多的机会达到 AUTs 的新状态。一定程度上缓解了基于模型的移动应用 GUI 的缺陷。

3.2.1 模型抽象和细化【2】

该方法不同于其他现有的基于模型的 GUI 静态测试方法，例如，模型在测试的过程中不会演化出他的抽象，因此常常是不准确的。该方法在测试运行的过程中，利用运行时信息来动态地优化模型，与现有方法相比，这种模型演化能力显著提高了模型的精度，从而显著提高了测试的有效性。

移动应用程序测试需要大量的人力投入。例如，测试人员编写代码来模拟 GUI 操作(例如单击一个按钮)来驱动 Android 应用程序的执行。这个过程不仅耗时，而且容易出错。此外，当 GUI 发生变化时，测试人员必须对他们现有的测试脚本进行重要的修改。模型为 GUI 测试提供了三种类型的好处。一、测试工具可以使用特定的遍历模型，系统地生成动作序列，然后回放动作序列，对 app 进行测试。其次，基于模型的测试工具生成由高级模型操作而不是低级事件组成的输入序列，这样可以方便回放。第三，可以将适当的抽象应用于模型，从而帮助减轻 GUI 操作的爆炸式增长。通过抽象，许多具有相同行为的 GUI 操作可以映射到相同的模型操作。由于这些 GUI 操作的行为是相同的，所以测试工具不需要测试每一个操作，而是可以在执行模型操作时，在其中选择一个有代表性的 GUI 操作。对状态进行抽

TABLE I: Model actions (*i.e.*, reduced attribute paths) by different abstractions.

Tree	Widget	Full Attribute Path	STOAT	AMOLA (C-Lv4)	AMOLA (C-Lv5)	Text-Only
T_i	w_0^i	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$
	w_1^i	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{XLSX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}0, \mathbf{t}{=}\texttt{XLSX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{XLSX}\}\rangle$
	w_2^i	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}1, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{PPTX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}1\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}1, \mathbf{t}{=}\texttt{PPTX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{PPTX}\}\rangle$
	w_3^i	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}2, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{DOCX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}2\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}2, \mathbf{t}{=}\texttt{DOCX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{DOCX}\}\rangle$
T_j	w_0^j	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}\rangle$	$\langle\{\mathbf{i}{=}0\}\rangle$
	w_1^j	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{DOCX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}0\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}0, \mathbf{t}{=}\texttt{DOCX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{DOCX}\}\rangle$
	w_2^j	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}1, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{XLSX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}1\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}1, \mathbf{t}{=}\texttt{XLSX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{XLSX}\}\rangle$
	w_3^j	$\langle\{\mathbf{i}{=}0, \mathbf{c}{=}\texttt{LV}, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}2, \mathbf{c}{=}\texttt{TV}, \mathbf{t}{=}\texttt{PPTX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \varnothing\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{i}{=}2\}\rangle$	$\langle\{\mathbf{i}{=}0, \mathbf{t}{=}\varnothing\}, \{\mathbf{i}{=}2, \mathbf{t}{=}\texttt{PPTX}\}\rangle$	$\langle\{\mathbf{i}{=}0\}, \{\mathbf{t}{=}\texttt{PPTX}\}\rangle$

* LV and TV are abbreviations for ListView and TextView, respectively.

象。将每个 GUI 操作映射到一个模型操作是状态抽象中最关键的步骤。因为具有相同模型动作集合的状态可以被合并，状态通常由它的模型动作集合来标识。

基于有效的动态模型抽象思想，该研究团队提出了一种新的、实用的基于模型的 Android 应用程序自动 GUI 测试技术——APE。一开始。用默认的抽象去初始化测试，这个

Algorithm 1: Model construction.

Input: Testing budget B, initial abstraction function $\mathcal{L}$.
Output: The model M.

```
(M, S, π) ← ((∅, ∅, ∅, L), ∅, ∅)                ▷ Initialization.
while B > 0 do
    B ← B − 1                                    ▷ Decrease the testing budget.
    T′ ← CaptureGUITree()                        ▷ Use uiautomator.
    M ← UptAndOptModel(M, S, π, T′)              ▷ See Algorithm 2.
    S ← L_L(T′)                                  ▷ Get the current state.
    π ← SelectAndSimulateAction(S)               ▷ See Section III-D.
return M
```

初始化的抽象可能是无效的，但是通过观察运行时的信息，APE 通过寻找更合适的抽象来细化模型。具体来说，APE 用一个决策树表示动态抽象，并使用测试期间获得的反馈动态地调整决策树。该方法和其他方法最主要的不同，就是有动态进化 GUI 模型的能力。

Algorithm 2: Update and optimize the model.

Function UptAndOptModel($M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{L}), S, \pi, T'$)
  Input: The new GUI tree T', the model $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{L})$, the previous state S, and the previous action π.
  Output: The new model.
  $S' \leftarrow \mathbb{L}_{\mathcal{L}}(T')$
  $M \leftarrow (\mathcal{S} \cup \{S'\}, \mathcal{A} \cup S', \mathcal{T} \cup \{(S, \pi, S')\}, \mathcal{L})$
  repeat $M \leftarrow$ ActionRefinement(M, T') **until** *M is not updated*
  $M \leftarrow$ StateCoarsening(M, T')
  return StateRefinement$(M, (S, \pi, S'))$

Function ActionRefinement($M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{L}), T'$)
  foreach $\pi' \in \mathbb{L}_{\mathcal{L}}(T')$ **do**
    if $|\overline{\mathcal{L}}(\pi') \cap T'| > \alpha$ **then**
      $R \leftarrow$ GetReducer(π')
      foreach $R' \in \{R' | R' \in \mathbb{R} \wedge R \not\sqsubset R'\}$ **do**
        $\mathcal{L}' \leftarrow \mathcal{L} \cup \{R \rightarrow R'\} \mid \mathcal{L} \cup \{(R, \pi', R')\}$
        $\Pi \leftarrow \{\mathcal{L}'(\sigma) | \sigma \in \overline{\mathcal{L}}(\pi') \cap T'\}$
        if $|\Pi| > 1$ **then**
          return RebuildModel$(M, \{\mathbb{L}_{\mathcal{L}}(T')\}, \mathcal{L}')$
  return M

Function StateCoarsening($M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{L}), T'$)
  $\mathcal{L}' \leftarrow$ GetPrev$(\mathcal{L})$
  $\mathbb{S} \leftarrow \{\mathbb{L}_{\mathcal{L}}(T) | T \in \overline{\mathbb{L}_{\mathcal{L}'}}(\mathbb{L}_{\mathcal{L}'}(T'))\}$
  if $|\mathbb{S}| > \beta$ **then return** RebuildModel$(M, \mathbb{S}, \mathcal{L}')$ **else return** M

Function StateRefinement($M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{L}), (S, \pi, S')$)
  foreach $(S, \pi, S'') \in \{(S, \pi, S'') | (S, \pi, S'') \in \mathcal{T} \wedge S'' \neq S'\}$ **do**
    foreach $\pi' \in S$ **do**
      $R \leftarrow$ GetReducer(π')
      foreach $R' \in \{R' | R' \in \mathbb{R} \wedge R \not\sqsubset R'\}$ **do**
        $\mathcal{L}' \leftarrow \mathcal{L} \cup \{R \rightarrow R'\} \mid \mathcal{L} \cup \{(R, \pi', R')\}$
        $\mathbb{S}_1 \leftarrow \{\mathbb{L}_{\mathcal{L}'}(T) | (T, \sigma, T') \in \overline{\mathfrak{L}_{\mathcal{L}'}}((S, \pi, S'))\}$
        $\mathbb{S}_2 \leftarrow \{\mathbb{L}_{\mathcal{L}'}(T) | (T, \sigma, T'') \in \overline{\mathfrak{L}_{\mathcal{L}'}}((S, \pi, S''))\}$
        if $\mathbb{S}_1 \cap \mathbb{S}_2 = \varnothing$ **then**
          return RebuildModel$(M, \mathbb{S}, \mathcal{L}')$
  return M

3.2.2 启发

通过动态调整决策树，缓解了一般基于模型的 GUI 测试的问题，即：工程量大，精确度不高，可移植性差等问题。

3.3.1 基于行为模型的 Android 应用随机测试【4】

主要解决思路是，创建一个可以被简单实用的，有着很小依赖和更高效率的自动化工具。依赖基于行为的 GUI 自动化测试方法，但是减少预先需要的设置初始化，并仍然能够达到相同或更好的覆盖率。因此，通过以下方法实现：

(1)通过在运行时映射 GUI 树和行为模型来避免静态分析

(2)动态更新模型，增加用户使用应用程序时很少或从不发生的事件发生的概率。

(3)应用统计模型创建行为模型。

虽然行为模型的方法并不新颖，但是此方法的独特之处在于，我们随着测试的进展不断地更新行为模型。最初的行为模型是使用统计建模从使用日志创建的，但是同时也考虑了没有出现在使用日志中的事件。这是通过将行为模型中的事件与 GUI 树中的 GUI 组件进行映射来实现的。事件发生的概率随着测试的进展而调整。

下图，是整个系统的架构图。

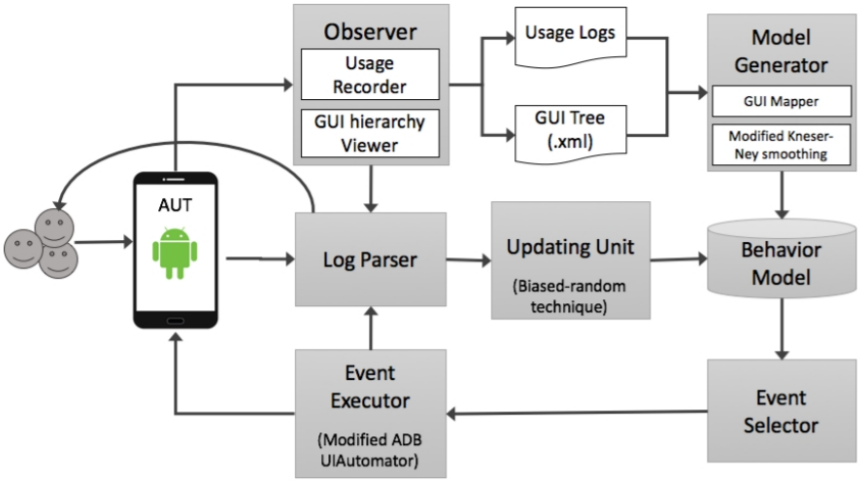


Fig. 1. An overview of the tool's architecture.

方法可以分为以下六个步骤:(1)事先收集使用日志，从 AUT 中提取 GUI 树(2)应用统计模型创建行为模型，然后与 GUI 树进行映射 (3)从行为模型中随机选择一个事件 (4)将选择的事件发射到 AUT (5)按频率调整概率，必要时更新模型 (6)重复第(2)至第(5)项，直至达到规定的时间限制或预先决定的次数。

在用 GUI 树映射行为模型的过程中，该方法并没有将事件和 GUI 组件放在一起。行为模型中的每个事件都被映射到 GUI 树中的一个 GUI 组件，以唯一的资源 id 作为键。

在选择阶段，会选择随机的事件，而不是选择发生概率高的事件。

$$e_{\text{next}} = \omega([P(e_1|h), P(e_2|h), P(e_3|h), \ldots, P(e_n|h)]),$$

在执行阶段，事件执行器执行下一个被选择的事件，工具可移植性类似点击，滑动等时间。

在更新阶段，执行阶段后，模型将将采用有偏随机技术去调整选中或执行的事件概率。他从前面选中并执行的事件中减少因素：

$$p'(e_x|h) = p(e_x|h) - d(e_x, \delta f_x),$$

在减少一些可能性后，要重新计算来维持一个合理的概率分布：

$$P(e_x|h) = \frac{p'(e_x|h)}{\sum_{i=1}^{n} p'(e_i|h)}.$$

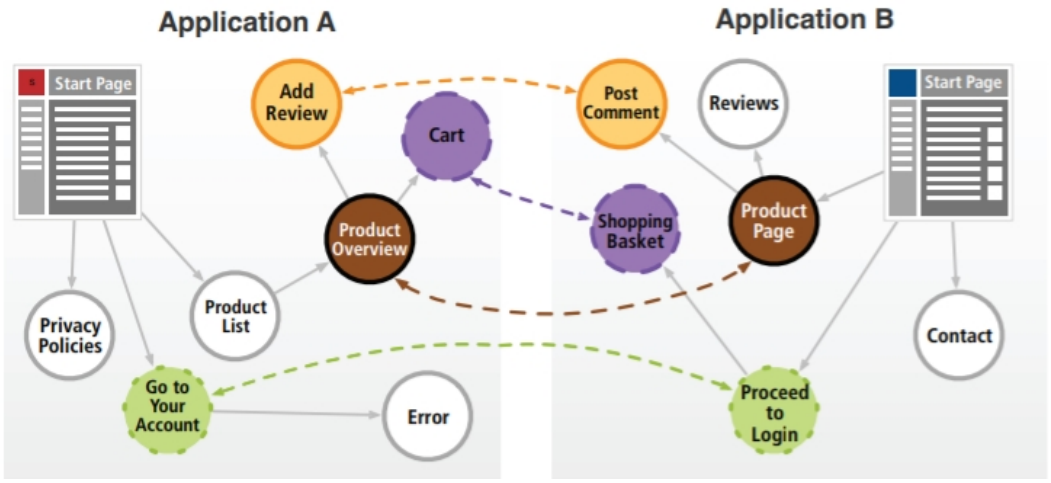
最后进行评估。

最终我们会发现，随意测试和认真测试可能会得到不同的结果。由于好处并不明显，我

们没有在录制期间提供特定的限制或目标。这会对此方法的有效性造成威胁。

3.4.1 通过学习其他应用的测试高效生成 GUI 测试【6】

为复杂的应用程序生成 GUI 测试非常困难，因为有很多功能需要去探索才能发现:例如，eBay 主页上显示了超过 2000 个单独的 GUI 元素，爬虫必须触发这些元素才能发现核心功能。该方法将展示如何利用其他应用程序的测试来指导新应用程序的测试生成。比如：对 Amazon 上的支付进行测试，我们可以引导 eBay 上的测试生成支付功能，利用跨两个应用程序的 UI 元素之间的语义相似性。在三个领域进行评估后，该方法允许在几个步骤中发现深层”功能，否则将需要成千上万的爬行交互。



利用跨应用程序的应用特性映射。利用语义文本相似度对两个应用程序的自然语言内容进行语义匹配。语义匹配就是该研究探究的事情。

在源程序测试中的每一步都会触发一个状态，从这些状态中抽象出语义特征，测试与之交互的用户界面元素及其文本标签。为了在目标程序中生成测试，要在探索目标程序的过程中想办法匹配这些特征，判断源程序和目标程序的语义相似性。最终的结果是将源程序中发现的特性映射到目标程序中发现的特性，从而使得那些为源程序创建的测试可以轻松适应新的目标程序。

文章举了一个例子形象说明了想法：如果我们曾经在 Amazon.com 这样的网站上购物过，我们就知道如何在其他购物网站上重复这个过程，这样我们就可以重新识别类似“购物车”、“结账”或“付款”这样的熟悉概念。简单明了。

3.5.1 基于图像的小部件检测的随机 GUI 测试改进方法【8】

这是一种通过使用机器学习，在屏幕快照中自动识别 GUI 小部件，来改进 GUI 测试的技术。

为了改进随机 GUI 测试，使用机器学习技术来识别屏幕快照中的小部件。挑战在于需要检索一个足够大的标记训练数据集，以支持应用流行的对象识别方法。生成这些数据的方法是:

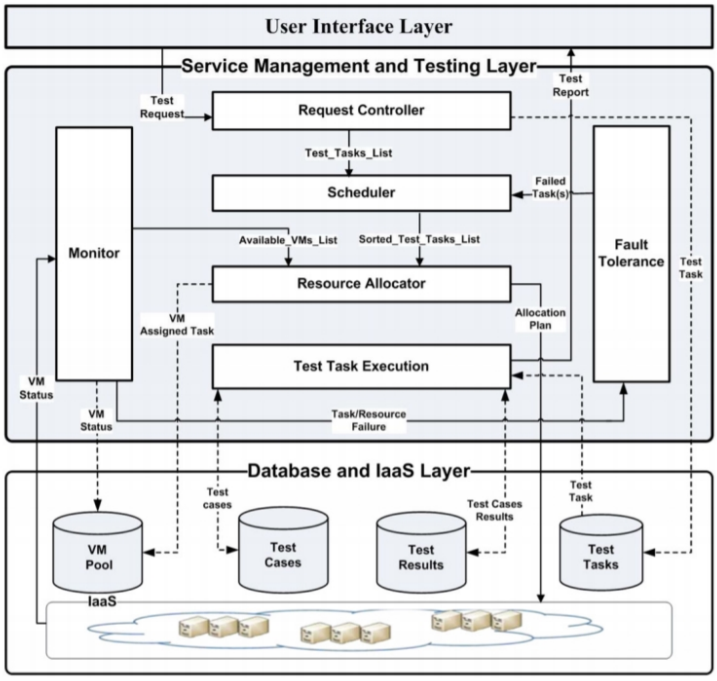
(1)生成随机的 Java Swing GUI

(2)使用小部件数据标记这些应用程序的屏幕快照。这些小部件数据是通过基于 Java Swing API 的 GUI 撷取来的。然后，经过训练的网络可以从测试生成期间的屏幕快照中预测小部件的位置和大小，从而影响 GUI 测试人员与应用程序的交互位置和方式。

3.6.1 自动并行 GUI 测试

GUI 测试不管是人工还是自动化，都需要很大的时间和努力。可以利用云计算技术克服传统软件工程领域的缺陷。因此，将测试作为一种服务，形成一种可以执行全部测试的模型，提出一种基于测试即服务体系结构的移动应用 GUI 测试系统。该系统执行所有测试活动，包括自动测试用例生成和在多个虚拟节点上并行测试。该系统减少了测试时间，满足了移动应用的快速上市约束。此外，所提出的系统架构解决了许多问题，如最大限度地利用资源，持续监控以确保系统的可靠性，以及应用容错方法来避免任何故障的发生等。

本文提出了一种面向 GUI 移动应用的 TaaS 体系结构。用户可以向建议的 TaaS 提交请求，整个测试过程将自动而有效地执行。提出的体系结构分为三层:(1)用户界面层，(2)服务管理与测试层，(3)数据库与基础设施即服务(Iaas)层。



4.1 移动应用 GUI 测试脆弱性研究

有证据表明，移动应用程序没有像桌面应用程序那样经过彻底的测试。特别是 GUI 测试通常是有限的。与基于 web 的应用程序一样，移动应用程序也受到 GUI 测试脆弱性的影响。GUI 测试在不改变应用程序功能的情况下，会由于 GUI 中的微小修改而失败。【5】【9】

正是因为移动应用测试的扩散性，当 GUI 进行简单修改后就可能造成失败，导致需要经常维护测试用例。因此，找到 GUI 测试脆弱性的原因，是改善 GUI 测试的重要前提和突破口。

GUI 测试的脆弱性可能存在以下原因：【10】

Table 5: Fragility cause occurrences

Fragility Cause	Layout-based	Visual
Text Change	5	3
Application Behaviour Change	2	2
Widget Substitution	2	0
Widget Removal	1	0
Graphic Change	0	5
Widget Arrangement	0	2
Widget Addition	0	1

1、 当测试用例因 GUI 元素的文本内容的更改而无效时，文本更改易碎性就会发生。

2、 图形更改的脆弱性是在同一个应用程序的两个连续发布版本之间对小部件的发生更改时造成的。

3、 当交互的小部件的类或类型在同一应用程序的两个连续版本之间发生更改时，会导致小部件替换的脆弱性。

4、 应用程序行为变化的脆弱性是由给定使用场景中应用程序的预期行为变化引起的。

5、 组件添加和组件移除的脆弱性是由于在 AUT 的 GUI 中的元素添加或移除引起的

5 总结和突发奇想

通过以上了解移动应用 GUI 测试的过程，我产生了一些想法。

不管是基于模型还是基于行为，不管是依赖机器学习还是其他方法，动态调整都是 GUI 测试很重要的一环。根据状态变化调整决策树等方法显著适应了 GUI 测试，并提高了 GUI 测试的可靠性。动态调整，应该在其他测试领域中也是不可或缺的一种方法。如果想要得到更加准确的测试结果，那么必定要依据程序执行状态动态调整以达到适应的结果。

对一些 GUI 测试方法，在不了解 GUI 小部件的情况下，测试生成工具只能盲目地与随机的屏幕位置交互。对这一盲目性，有机器学习方法去改善，还有其他的基于模型、行为等的解决方案，但是这些解决方案都是脱离源程序，作为测试方法独立进行的，并且准确性难以得到显著的改善。

对于 GUI 测试独立于开发，我认为有好有坏。好在于测试人员可以专注于测试，开发人员可以专注与开发，二者互不影响（GUI 测试主要在前端），但是坏处就在于，二者分离，测试人员获取组件信息的途径狭隘。

我认为如果有可能的话，可以开发一种组件注册工具，在前段开发时，就讲这些组件注册，并记录行为，具体的衡量标准、注册内容我并没有构思好。但是如果就前端开发而言，在我们声明一个组件的时候，添加动作，样式等附加时，如果直接自动注册到一个组件注册表中，不仅可以 GUI 测试人员进行 GUI 测试，且自动化难度降低，而且由于组件自动注册，不影响前端人员开发过程，对前端人员来说，注册的过程是透明的。

如果想要提升 GUI 测试的可靠性和准确性，可以在如何完善注册表注册内容上努力，已达到改善目标。这只是我的一个不成熟的突发奇想。

虽然在敏捷开发中，简单的测试任务交由开发人员完成，测试和开发在分离开来后又进行了简单的融合。我认为测试和开发的界限被我们过度限定了。其实开发的一侧，可以做一些努力，来便捷测试；测试的一侧也应该可以做一些大胆的想法，让开发更加便捷。

开发的过程中生成组件注册表，我认为就是开发的一侧在 GUI 测试中对便捷测试的一侧的一个可行方案。请老师批评指正！

参考：

【1】 Yuzhong Cao ， Guoquan Wu ， Wei Chen ， Jun Wei “CrawlDroid: Effective Model-based GUI Testing of Android Apps” Proceedings of the Tenth Asia-Pacific Symposium on Internetware

【2】 Tianxiao Gu，Chengnian Sun ，Xiaoxing Ma ，Chun Cao ，Chang Xu ，Yuan Yao Qirun Zhang ， Jian Lu ， Zhendong Su “Practical GUI Testing of Android Applications via Model Abstraction and Refinement” 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

【3】 Hendrik Bünder ，Herbert Kuchen“A Model-Driven Approach for Behavior-Driven GUI Testing” ， Proceedings of the 34th ACM/SIGAPP Symposium on Applied

Computing

【4】 Woramet Muangsiri and Shingo Takada† “Random GUI Testing of Android Application Using Behavioral Model” International Journal of Software Engineering and Knowledge EngineeringVol. 27, No. 09n10, pp. 1603-1612 (2017)Research Notes

【5】 Riccardo Coppola, Maurizio Morisio and Marco Torchiano “Scripted GUI Testing of Android Apps: A Study on Diffusion, Evolution and Fragility” Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering , ACM

【6】 Andreas Rau , Jenny Hotzkow , Andreas Zeller “Poster: Efficient GUI Test Generation by Learning from Tests of Other Apps”2018 ACM/IEEE 40th International Conference on Software Engineering: Companion Proceedings

【7】 Thomas D. White , Gordon Fraser , Guy J. Brown “Improving Random GUI Testing with Image-Based Widget Detection” Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis

【8】 Amira Ali , Huda Amin Maghawry , Nagwa Badr “Automated parallel GUI testing as a service for mobile applications” preceding in Software:Evolution and process October 2018

【9】 Kochhar PS, Thung F, Nagappan N, Zimmermann T, Lo D.“ Understanding the test automation culture of app developers.” in software testing, verification and validation (ICST), 2015 IEEE 8th international conference on, pp. 1-10. IEEE, 2015.

【10】 Riccardo Coppola , Luca Ardito , Marco Torchiano “Fragility of Layout-Based and Visual GUI Test Scripts: An Assessment Study on a Hybrid Mobile Application”