论 APP UI 自动化测试的可行性

背景

在这个科技时代，app数量也是逐年递增，只要能想到，大多数都可以在相应的平台找到相关的app!

那一款app在市场上出现之前，会经过哪些“加工锻造”的过程呢？又经过哪些流程，来确保从市场上下载下来的app能正常使用呢？在这里我，以测试的视角来说我对app一些粗浅认知,在这过程一些意见，纯属个人臆断，如有不对欢迎指正！

不管是何种事物的产生，都是基于需求，有需求才能存在发展的内在驱动力，因此app的产生的第一阶段是需求；有了需求就要基于该需求，调查市场上的用户的群体，分析获取用户的功能性需求，设计app的功能性需求，因此这一阶段为产品设计；根据产品的文档，开始app的制作加工阶段，这个阶段不展开解说，包括内容太多，例如开发框架的选择、UI设计、编码等等，这个阶段为app设计与实现阶段；根据app设计与实现完成，要确保能正常使用，需要进行相应的测试，该阶段称为app测试阶段；测试完成之后，app上线，至此一个app的产生就完成了，至于市场运营，那是以后的事情了！

因此，app的加工锻造的过程，大致为：获取市场需求、提取需求设计产品文档、根据需求设计并实现app、app测试、app上线！

本篇文章，是针对app测试阶段中的UI自动化测试的可实施性进行分析作为背景，从需求、技术、维护三个阶段来分析UI自动化测试在app的可行性！

需求变更频繁

己的能力增长那是很大的不是吗？就以我遇到的一些问题为例，每个app产生的时候，都有自己的一些核心业务，后期的版本更新都是基于这些核心业务来扩展设计新的功能，这些核心业务基本很少甚至不发生变化，那么能不能对于app中成熟的业务来设计UI自动化测试用例呢？那么对于这个想法，我做了如下尝试，把遇到坑归纳如下：

核心业务是否有单独稳定的UI入口？

核心业务是否有单独稳定的UI入口，这是重中之重，这能保证测试的正常进行，自动化测试的初期，都是对业务流进行梳理分析，选择一些成熟稳定的业务流！这么做的好处主要有：需求变更对业务流影响较少、有效的数据比较多、业务之间的边界清晰！

如果核心业务没有单独稳定的入口，那么UI的自动化还没开始就会遭遇滑铁卢，因为入口变更频繁的话的，那么业务流就无法成功走下去，后续的测试用例也无法成功执行！其实测试做到后期，都有个认识，UI自动化测试都是基于业务流的测试，业务流发生变化，直接影响的是一个测试集的，因此针对APP的UI自动化测试而言，业务流的选择是基石！

一些开发的编码格式是否规范？

开发人员的编码规范与否，对于自动化测试的也是有一定影响的，规范的编码，对于后期的技术实现影响是相当大的，所以早期，最好能与开发协商，你测试过程中需要的一些元素属性一定不能为空，方便后期元素定位，因此开发人员编码规范也是相当重要的！

App开发框架更新？

APP开发框架一般对于业务流影响不大，可能会影响部分测试集中的测试用例，导致测试用例无法正确执行，这个影响程度要看测试用例的数量，因此这个对UI自动化影响大小看情况而定！

因此，根据上述可以看出，APP UI自动化测试可行性在一些场合下是可行，在一些场合

环境部署

针对Appium环境部署，这里不讲如何安装配置，如果想学习Appium+Python在Windows下面的安装，这里给我之前的一篇博客链接：Appium+Python在Windows下面的安装

这个安装方法，我试过，是可以成功安装，如果在安装过程遇到一些问题，可以在该博客下留言，我会及时回复的！本人在刚开始，配置部署这个开发环境时，就踩了好多坑，环境配置是我目前最麻烦的之一，比Spark集群稍微简单点，但是它的需求太多，好多想要使用该工具的人，估计90%的人被环境配置难住！

环境配置结束之后，那就是需要跑一个测试，来了解下简单的操作流程。本文不做赘述，链接：**appium简单实例**

示例代码：

```
    import time
    desired_caps = {}
    desired_caps['deviceName'] = 'GT-N7100'
    desired_caps['platformName'] = 'Android'
    desired_caps['browserName'] = ''
    desired_caps['version'] = '4.3'
    desired_caps['appPackage'] = 'com.job.android'
    driver = webdriver.Remote('http://127.0.0.1:4723/wd/hub',
desired_caps)
    time.sleep(5)

driver.find_element_by_id("com.job.android:id/closebtn").click()
    time.sleep(10)

driver.find_element_by_id("com.job.android:id/tv_msg_remind_right
").click()
    time.sleep(10)

driver.find_element_by_id("com.job.android:id/loginbutton").click
()
```

- **find_element_by_class_name()**：根据元素的class属性值进行定位；
- **find_element_by_tag_name()**：根据元素的tag名进行定位；
- **find_element_by_link_text()**：根据链接文本进行定位；
- **find_element_by_partial_link_text()**：根据链接文本中包含的部分文本信息进行定位
- **find_element_by_xpath()**：根据xpath进行定位；
- **find_element_by_css_selector()**：根据css进行定位；

关于定位元素组的，只需要把find_element改成find_elements就可以。上述是selenium的定位方法，目前IOS使用XCUITesting后，元素定位方法与selenium元素定位方法一致。这是Appium+XCUITest的Python链接：

Appium+XCUITest基于Python的操作实例以及环境搭建

那么appium如何在安卓设备上进行定位呢？在安卓设备上使用的是uiautomator，以Python为例，使用方法如下：find_element_by_android_uiautomator()，该方法与Uiautomator的UiSelector()联合使用来定位元素。下面给出一段代码示例截图：

```
#coding=utf8
from time import sleep
from appium import webdriver
from desiredCaps import setDesired_caps

class appDriver():
    def __init__(self,host='http://localhost:4723/wd/hub'):
        try:
            desired_caps =setDesired_caps().desired_caps
            self.appdriver= webdriver.Remote(host,desired_caps)
        except Exception,e:
            print "Driver Error",e
    #根据“控件text属性的内容”构造出UiSelector对象
    def app_find_element_by_text(self,text):
        try:
            item='new UiSelector().text("'+text+'")'
            #进行费时装换，如果不进行格式这步格式转换，会出现HTTP错误
            item="%s"%(item)
            element=self.appdriver.find_element_by_android_uiautomator(item)
            return element
            self.appdriver.f
        except Exception,e:
```

可以查看我的博客链接：**python封装安卓查找元素方法V1.0**

性能

appium对设备的要求比较高，如果设备配置的比价低的，运行脚本时会相当卡顿，不能及时响应操作，总体来说性能不怎么好！

使用难度

从环境的配置，到编码的使用，都要求较高，需要有一定的编码基础，不然很难入门使用，所需要花费的成本较高！不过可以与robot framework结合使用，使用者不需要关心后台代码逻辑！关于如何与robot framework可以从网上查找，比较多的！

针对上述可以发现，app UI自动化测试上技术是可行的，不过对于测试人员的技术要求比较高，学习成本比较大！需要专业的人员指导，才能顺利实施！

关于在windows下对安卓进行的自动化测试，我做了做了一个研究，把adb+python结合起来，把adb的命令和Python结合起来，来实现元素点击操作，具体代码链接：**通过adb与python结合创建的设备驱动脚本deviceDriver.py**

部分代码截图：

```
        print "Get the list of  the  position of elements Error!"

    def findElementByName(self, name):
        """
        通过元素名称定位
        usage: findElementByName(u"设置")
        """
        return self.__element("text", name)

    def findElementsByName(self, name):
        return self.__elements("text", name)

    def findElementByClass(self, className):
        """
        通过元素类名定位
        usage: findElementByClass("android.widget.TextView")
        """
        return self.__element("class", className)

    def findElementsByClass(self, className):
        return self.__elements("class", className)

    def findElementById(self, Id):
        """
        通过元素的resource-id定位
        usage: findElementsById("com.android.deskclock:id/imageview")
        """
        return self.__element("resource-id",Id)

    def findElementsById(self, Id):
        return self.__elements("resource-id",Id)

    #点击元素
    def ClickElement(self,dx,dy):
        try:
            if self.devices:
                tap=os.popen("adb shell input tap " + str(dx) + " " + str(dy))
            else:
```


改脚本实施原理是，通过adb命令把设备当前页面的信息dump下来保存为xml文件，通过xml的树定位元素在页面所在的位置，具体可以看上述代码链接，具体实施是可行的

脚本维护，是保存自动化测试持续的必要条件，人员流失、文档少，是自动化测试脚本废弃的主要问题！

总结

其实，在研究app UI自动化测试可行性分析，发现最大的问题是：**入口和UI经常变化、时间紧，导致维护成本高**！其实，app UI自动化脚本开发都是提高工作质量，但是由于app更新频次快，时间紧，手动测试比较自由及时，使用脚本还要去调试校验、编写代码，比较耗时！

总而言之，APP UI自动化因公司而定，如果有规范化的管理流程，自动化测试在回归测试方面还是很高效的！