1. 获取用户操作流，有两种获取方式，可见相应文件夹目录下的说明。得到后，将文件放置于项目根目录下的OperationFlow/{app包名} 文件夹下即可。
2. 使用要求：java1.8,安卓sdk（至少把aapt配到系统变量里去能直接调用了），appium要求客户端在1.9以上，node:v8.12.0
3. 运行方式分两种，第一种是融合到了平台中，可见<http://git.mooctest.net/HenryLee/androidAutoTest/tree/MergeScriptTest>，此处为直接对学长的原先的初始算法文件进行了替换，因此按照学长平台的使用方法使用即可。该方法产生的结果比较统一且有条理性。
4. 第二种是实验环境，直接使用ColinTest整个项目，只做了从idea中启动的适配，没搞生成jar包。运行见mergeTest/Driver.java中的runInIDE方法，其中为运行配置。mID指这台设备的UDID，apkname啥的就不说了。（至于为啥只加了一个手机，因为加多个的那块已经在学长那实现了……这里只是一个实验环境，想加也行，直接向list里add就行）
5. 具体的执行流程就跟着ide一路看一下就懂了。会为每个连接的设备申请MergeTest类，这个类包含的内容实际上是指一个测试的task，会运行类里的startTestTask方法返回相关的task信息。 至于这个方法里面做了啥，可以看该类的run方法。
6. 原本工具自己跑的逻辑没变，主要是改了一下跑之前的部分，把所有的自动化内容和学习脚本的内容全部放在executeScriptWithHumanKnowledge方法里了，里面有注释，就不再赘述了。
7. 其余涉及到的和用户操作流有关的代码全部放在AppiumNode包里，并不是所有的部分都用到了，但是都有写注释，所以要改起来应该也挺方便。
8. 具体的跑的流程就是： 读用户操作流文件->通过对文件分析和融合，得知目前应用已经被探索的控件，和这些控件之间相连的信息->工具跑一编->找已知信息里还没覆盖到的控件（且所在activity也没到过）->由于整个系统基于appium，因此轻易的复现到该控件处->让工具继续跑->循环直到跑完。 此处如果想再多点覆盖率，可以让其他处于覆盖过的activity上的，没覆盖过的控件也加入循环，多跑N遍，实验结果表明确实能提高覆盖率，不过时间也会大大增加就是了……
9. 工具自带控件检查，因为在跑的过程中自己也会生成应用模型，所以多次循环后如果到了跑过的页面不会傻傻的再跑一次。
10. 其余的内容应该基本都是这个祖传项目里原本就有的，比如Appium建立等等。很多内容都是通过执行sh/bat脚本来执行的，用之前最好检查下自己的系统。实验环境下的这个是windows没问题，linux没尝试，学长那个是windowx没有适配，只有mac。。