卓有成效的移动 App 系统测试

The Effective System Testing of Mobile App

目 录

1. 简介 3

1.1 什么是 App 测试 3
1.2 测试方法 4
1.2.1 白盒测试 4
1.2.2 黑盒测试 4
1.2.3 人工测试 4
1.2.4 自动化测试 4
1.3 UT、IT、ST 测试 4
1.3.1 Unit Testing 单元测试 4
1.3.2 Integrate Testing 集成测试 4
1.3.3 System Testing 系统测试 4

2. 移动 APP 的系统测试 5

2.1 冒烟测试（Smoke Testing） 6
2.2 功能测试（Functional Testing） 6
2.3 用户界面测试（GUI Testing） 6
2.4 用户体验可用性测试（UE Testing） 7
2.5 安全性、访问控制测试（Security Testing） 7
2.6 性能测试（Performance Testing） 8
2.6.1 负载测试（Load Testing） 8
2.6.2 强度测试（Stress Testing） 8
2.6.3 稳定性测试（Stability Testing） 8
2.6.4 基准测试 9
2.6.5 竞争测试 9
2.7 故障转移和恢复测试（Recovery Testing） 9
2.8 兼容适配性测试（配置测试 Configuration Testing） 9
2.9 分辨率测试 10
2.10 网络测试 10
2.11 本地化测试 11
2.12 文字测试 11
2.13 发布测试 11
2.13.1 说明书测试 12
2.13.2 宣传材料测试 12
2.13.3 帮助文件测试 12
2.14 回归测试 12

1. 简介

移动应用 App 已经渗透到每个人的生活、娱乐、学习、工作当中，令人激动、兴奋且具有创造性的各种 App 犹如雨后春笋般交付到用户手中。各类智能终端也在快速发布，而开发者对于全球移动设备的质量和性能却掌握甚少，App 与设备的兼容性问题常常导致用户投诉，令开发者十分沮丧，App 测试与服务质量保证矛盾十分突出。

移动开发的一个重要难题，就是应用在开发过程中，必须使用手机真实环境进行系统测试，才有可能进入商用。由于手机操作系统的不同，以及操作系统版本之间的差异，使得真机系统测试这个过程尤其复杂，涉及终端、人员、工具、时间、管理等方面的问题。

首先必须购买足够多的手机，包括不同操作系统，不同版本，不同分辨率，甚至不同厂商，目前市场上的手机平台有 iOS、Android、Symbian、WP、Blackberry、Linux 等（集中度较高的是 iOS、Android 和 WP 系统），平台之间存在较大差异，语言和标准完全不同。以 Android 为例，就需要面对 Android 1.5、2.0、2.1、2.2、2.3、3.0、4.0 七个以上的版本，约十几种不同的分辨率，HTC、摩托、三星、LG、索爱、联想、中兴、华为…等数十个厂商。一个商业化运作的开发团队，一般至少需要几十部手机、终端，才能完成必要的适配工作。如果缺失这个真机系统测试环节，极大可能会给应用的推广和使用埋下了一个隐患，一旦出问题将直接招致用户的投诉或抛弃。

其次在拿到不同手机进行测试的时候，还将面临不同手机厂商的系统版本差异问题，即便是标准统一的 Android 系统，手机厂商的版本也并非完全相同，MIUI、LePhone、MEIZU，这些 Android 系统已经加入了很多个性化的东西，导致 Android 应用必须进行单独适配。这过程中出现的很多问题，往往没有资料可查，使开发者雪上加霜。

终端问题之后，就是人员工资的高涨使得很多开发团队在紧张的预算下优先向产品、运营、技术倾斜，很多成规模的互联网企业通常只有几个人的小测试团队。

另外，App 的真机系统测试在全球范围内还停留在刀耕火种的纯人工状态，没有有效的工具可以利用，测试人员发现的 Bug 很难复现，开发人员因此也很难定位、快速修改 Bug。

接下来的问题是，为了满足用户旺盛的需求、适应激烈的市场竞争，所有的移动互联网企业都在拼命地赶工期，开发人员下班前完成的版本、至少希望第二天上班的时候能够被测试完成，这就要求测试人员连夜工作，于是我们可以看到很多欧美的软件公司会把测试工作交给中国的外包企业进行。

最后，终端、人员、流程等管理问题也非常突出，终端、Bug、人员要在测试、开发、产品、客服、运营等不同的部门之前交错。

如何进行卓有成效的 App 系统测试，以及协调好与之相关的计划、管理、人员、资源、终端等各个环节，一直是困扰各个 App 开发企业的问题。

1.1 什么是 App 测试

IEEE 定义：使用人工或自动化来测试某个程序，来验证它是否满足规定的需求或者实际结果和预期结果之间的差别。

App 是基于移动互联网软件、及软硬件环境的应用软件。App 测试就是要找出 App 中的 BUG，通过各种手段和测试工具，判断 App 系统是否能够满足预期标准。移动 App，由于增加了终端、外设和网络等多项元素，因而测试内容和项目也相应增加了。

在 App 开发过程中容易出现缺乏有效沟通，功能复杂、编程错误、需求不断变更、时间压力、缺乏文档的代码、App 开发工具、SDK 和人员的疏忽等原因引发的错误，通过测试能够发现、找出其中的错误，解决错误，从而提高 App 的质量。

1.2 测试方法

1.2.1 白盒测试

依据被测 App 分析程序内部构造，并根据内部构造设计用例，来对内部控制流程进行测试。

1.2.2 黑盒测试

黑盒测试（Black-Box Testing）是基于系统需求规格，在不知道系统或组件的内部结构的情况下进行的测试，把测试对象看作一个黑盒，只考虑整体特性，不考虑内部具体实现。

通常又将黑盒测试叫做：基于规格的测试（Specification-Based Testing）、输入输出测试（Input/Output Testing）、功能测试（Functional Testing）。

1.2.3 人工测试

测试活动由人来完成，狭义上指测试执行由人工完成。

1.2.4 自动化测试

通过计算机模拟人的测试行为，替代人的测试活动，狭义上指测试执行由计算机来完成。

1.3 UT、IT、ST 测试

1.3.1 Unit Testing 单元测试

定义：对 App 的基本组成单元来进行正确性检验。集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。

目的：检测 App 模块对 App 产品设计说明书的符合程度。

类型：白盒测试，测试范围为单元内部的数据结构，逻辑控制，异常处理。

评估标准：逻辑覆盖率。

1.3.2 Integrate Testing 集成测试

定义：测试模块或子系统组装后功能以及模块间接口是否正确，把已测试过的模块组装起来，主要对与设计相关的 App 体系结构的构造进行测试。

目的：在于检测 App 模块对 App 产品概要设计说明书的符合程度。

类型：灰盒测试，测试范围为模块之间接口与接口数据传递的关系，以及模块组合后的功能。

评估标准：接口覆盖率。

1.3.3 System Testing 系统测试

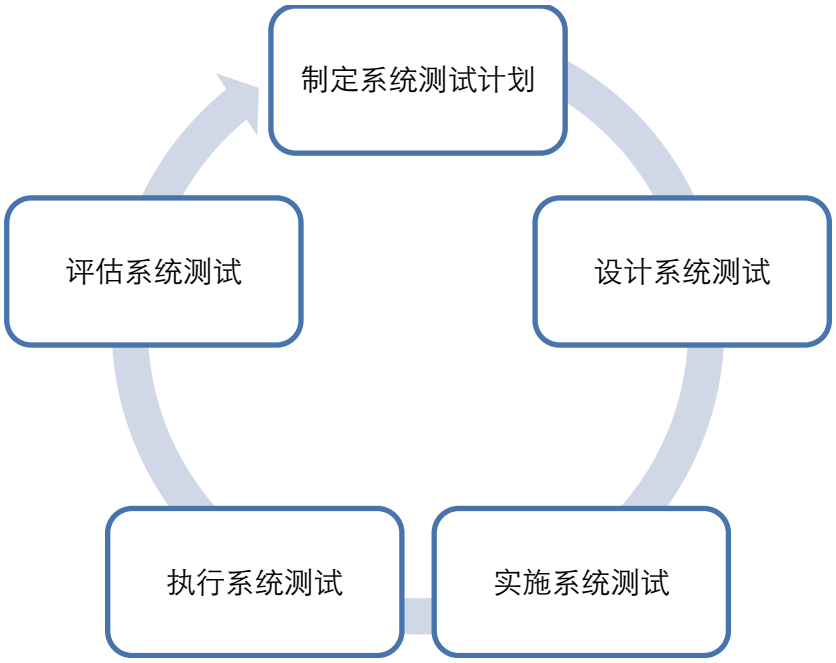
定义：App 系统测试（App System Testing），是将已经确认的 App 程序、移动终端、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试，系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不符或与之矛盾的地方，从而提出更加完善的方案。App 系统测试发现问题之后要经过调试找出错误原因和位置，然后进行改正。

App 系统测试是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。对象不仅仅包括需测试的 App 软件，还要包含 App 软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等，基于本地及不同地区、网络等真实终端，测试、检查已实现的 App 是否满足了需求规格说明中确定了的各种需求，以及 App 配置是否完全、正确。

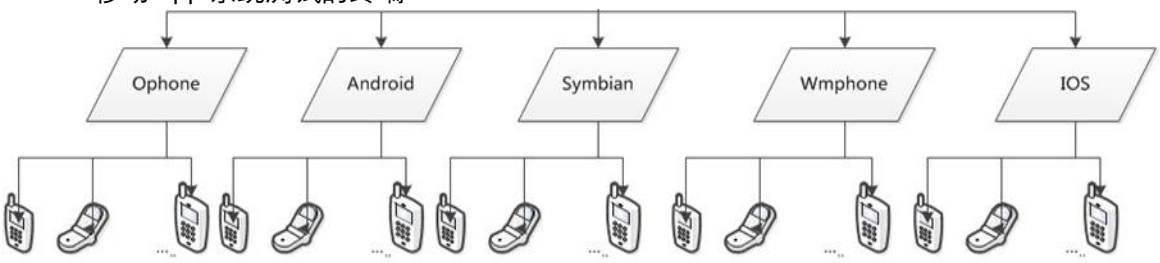
目的：验证最终 App 系统是否满足用户规定的需求。

类型：黑盒测试，测试范围为整个系统。

评估标准：测试用例对需求规格的覆盖率。
系统测试过程：



移动 App 系统测试的终端：



2. 移动 App 的系统测试

目前主流的 iOS、Android 和 WP 等 OS 系统以及各平台，都相应地提供了不同程度的单元、集成测试工具，可以在模拟器、沙箱环境下进行白盒、灰盒测试、调试。

但 App 存在着大量的软硬件交互，而这些都需要通过真实的终端通过黑盒测试方法进行系统测试，需要将经过集成测试的软件，作为移动终端的一个部分，与系统中其他部分结合起来，在实际运行环境下对移动终端系统进行一系列严格有效地测试，以发现软件潜在的问题，保证系统的正常运行，验证最终软件系统是否满足用户规定的需求。

然而，由于 OS 版本、硬件异常迅猛的发展速度，平台始终没有有效地提供符合 App 黑盒系统测试的工具与方法，大量的移动 App 测试还停留在纯人工状态，效率十分低下。终端、版本的碎片化，更加剧了这一问题的严重性。

自己开发、或借助第三方工具、平台，进行自动化的移动互联网 App 系统黑盒测试，是提升效率和测试质量的有效方法。

移动互联网是极快速发展的新兴产业，没有成功经验可循，只有市场和用户才是检验你产品是否好坏的终极标准。借助传统软件测试方法和规律，可以有效地提升 App 的程序质量和用户体验。

2.1 冒烟测试（Smoke Testing）

冒烟测试（Smoke Testing）的对象是每一个新编译的需要正式测试的 App 版本，目的是确认软件基本功能正常，可进行后续的正式测试工作。冒烟测试的执行者是版本编译人员。

App 程序在编写开发过程中，内部需要多个版本(Builds)，但是只有有限的几个版本需要执行正式测试（根据项目开发计划），这些需要执行的中间测试版本。在刚刚编译出来后，开发人员需要进行基本性能确认测试，验证 App 是否能正确安装、卸载，以及操作过程和操作前后对系统资源的使用情况，针对终端硬件及 ROM 版本的各维度，与 App 安装、卸载不适配情况、隐患原因分析报告，最终确认是否可以正确安装/卸载，主要功能是否实现，是否存在严重死机、意外崩溃等 Bug。

如果通过了该测试，则可以根据正式测试文档进行正式测试。否则，就需要重新编译版本，再次执行版本可接收确认测试，直到成功。

如果发现问题，就要有效地发现导致问题出现的原因，例如在 Android App 测试中，某些终端、有时会出现应用程序错误需要强行关闭的提示，但又找不到重现这个问题的步骤，这个是 App 的问题还是系统的问题呢，应该怎么判断呢？这通常需要有 Log 日志才可以判断，Andriod App 出现 Crash 的情况，一般有两方面的原因，如果 Log 日志中出现 System_server，则为系统问题；如果 Log 中出现 Shutdown VM，代表应用程序的问题；还有一种情况是出现 Died，这个是进程死掉导致，包含系统主动杀死的情况。

【Testin Tips】

当一个单元、或程序整体开发编译完成，开发人员、或测试人员可以在 PC 上选取被测的 App，通过 iTestin 连接的原型测试终端，自动进行快速的冒烟测试，以验证 App 安装、启动、基本操作运行、卸载等是否正常，测试报告包括各测试项是否成功、特征截图、Log 日志、CPU/内存等参数等。

2.2 功能测试（Functional Testing）

功能测试是移动 App 测试最关键的环节，根据产品的需求规格说明书和测试需求列表，验证产品的功能实现是否符合产品需求规格；

功能测试的目标主要包括：

- ✓ 是否有遗漏需求；
- ✓ 是否正确的实现所有功能；
- ✓ 隐示需求在系统是否实现；
- ✓ 输入、输出是否正确。

移动 App 的功能测试应侧重于所有可直接追踪到用例、或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。

功能测试基于黑盒技术，通过图形用户界面（GUI）与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。

【Testin Tips】

通过 iTestin Suite 连接的本地终端，测试人员可以非常方便地按照测试用例、在终端上进行操作，所有的操作过程、轨迹都会被自动记录为脚本，所有操作目标、特征点的屏幕截图、Log 日志、CPU/内存/网络和其他系统资源参数也都会被详细的记录下来，最后形成测试报告。

当功能测试要求涉及到不同地区、不同网络的时候，可以发布任务到 mTestin 社区，要求特定地区、特定网络的测试者按照功能测试用例要求进行测试，然后通过报告汇总测试结果。

2.3 用户界面测试（GUI Testing）

用户界面（GUI）测试用于核实用户与 App 之间的交互，包括用户友好性，人性化测试。

一个好的 App 要有一个极佳的分辨率，而在其他分辨率下也都能可以运行。GUI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外，GUI 测试还可确

保 GUI 中的对象按照预期的方式运行，并符合公司或行业的标准。

GUI 测试主要测试在不同分辨率下，测试用户界面(如菜单、对话框、窗口和其它可视控件)布局、风格是否满足客户要求，文字是否正确，页面是否美观，文字，图片组合是否完美，操作是否友好等。

GUI 测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。确保用户界面符合公司或行业的标准，包括用户友好性、人性化、易操作性测试。

【Testin Tips】

通过 iTestin Suite 完成冒烟测试和功能测试，所有特征点的截图都可以快速反馈 UI 是否正常。

同时，为了更好地测试普通用户对 UI 的反馈，可以进行用户 UI 测试，找一组人(1 组 12 人，军队一个班的建制)，试用你的原型 UI，记录他们的操作轨迹，当然包括严重的 Bug：

1） 邀请外部用户在现场通过 iTestin Suite 连接的终端进行操作。

2） 发布测试任务到 mTestin 社区，要求 n 组用户按照 UI 测试要求，通过用户自己的终端连接到 iTestin 进行操作，将完成的任务提交到 mTestin 社区。

通过此类测试，可以有效地发现不同用户操作 UI 上的行为轨迹差异，以判断 UI 是否存在设计不恰当、或许要改进的地方。

2.4 用户体验可用性测试（UE Testing）

用户体验可用性测试主要是检测用户在理解和使用系统方面到底有多好，是否存在障碍或难以理解的部分。

用户体验可用性的测试方法，一般是通过用户访谈，或邀请内测、小范围公测等方式进行，通过不同实验组的运营结果来判断是否存在可用性缺陷。但由于缺乏有效的测试工具，必须要大量的测试样本才能获得比较真实的测试数据，投入资源较多，测试周期较长。

【Testin Tips】

参考 GUI 测试方法，为了更好地测试普通用户对 App UE 的反馈，可以进行用户可用性测试，找 n 组测试者（1 组 12 人——军队一个班的建制，UE 测试建议选取最大 12 组测试者、144 人），试用 App 的原型 UE 可用性，记录测试者的操作轨迹，当然包括严重的 Bug。

1） 邀请外部用户在现场通过 iTestin Suite 连接的终端进行操作。

2） 发布测试任务到 mTestin 社区，要求 n 组用户按照可用性测试要求，通过用户自己的终端连接到 iTestin 进行操作，将完成的任务提交到 mTestin 社区。

通过此类测试，可以有效地发现不同用户操作 App UE 行为轨迹差异，以判断 App 的 UE 是否存在设计不恰当、或许要改进的地方。

2.5 安全性、访问控制测试（Security Testing）

安全性和访问控制测试侧重于安全性的两个关键方面：

1) 应用程序级别的安全性，包括对数据或业务功能的访问。

应用程序级别的安全性可确保：在预期的安全性情况下，主角只能访问特定的功能或用例，或者只能访问有限的数据。例如，可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性，测试就可确保“用户类型一” 能够看到所有客户消息（包括财务数据），而“用户二”只能看见同一客户的统计数据。

2) 系统级别的安全性，包括对系统的登录或远程访问。

系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。

【Testin Tips】

由于 App 安全性、访问控制测试，通常需要设定不同的用户使用场景和多种变化事件，非常耗费人力、时间与资源。比较有效的执行方法，是将 App 安全性、访问控制测试任务进行分解，发布到 mTestin 进行群测，这样可以快速、成本可控地通过广泛的社会化专业测试人员的有偿参与，完成测试任务。

由于 mTestin 群测的测试者的测试过程，会被全程通过 iTestin 进行记录，监控、稽核测试结果的完整性与有效性。

2.6 性能测试（Performance Testing）

性能测试用来测试 App 在真实环境中的运行性能，以及与硬件、网络资源的匹配度，最终度量系统相对于预定义目标的差距，通过极限测试方法，发现系统在极限或恶劣的环境中自我保护能力，主要验证系统的可靠性。

性能测试测试主要通过以下几项测试完成。

2.6.1 负载测试（Load Testing）

负载测试是在一定的软硬件及网络环境下，通过模拟不同的用户，执行一种或多种业务，观察系统在不同负载下的性能表现。在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。

负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。

此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

【Testin Tips】

性能测试可以通过 iTestin 录制脚本，在本地的原型终端上单机进行，也可以在 iTestin 组成的企业私有云、或 Testin 终端云上发起任务。

2.6.2 强度测试（Stress Testing）

强度测试是一种性能测试，实施和执行此类测试的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足，测试对象就可能会表现出一些在正常条件下并不明显的缺陷。而其他缺陷则可能由于争用共享资源（如数据库锁或网络带宽）而造成的。强度测试还可用于确定测试对象能够处理的最大工作量。

【Testin Tips】

强度测试可以根据测试要求，通过 iTestin 录制脚本，本地、或通过企业私有云、甚至 Testin 公有云的终端，非常方便、有效地设定多次执行的次数，自动进行测试，例如选 99 件商品、加 999 个好友、上传 9999 张照片、支付 99999 次等等。

2.6.3 稳定性测试（Stability Testing）

稳定性测试评价系统在一定负荷情况下，长时间的运行情况。在一定的软硬件及网络环境中，通过模拟大量的用户执行多种业务处理大量数据，使系统在极限环境下长时间运行，目的在于寻找系统的失效点。

【Testin Tips】

稳定性测试可以根据 App 的产品特征，非常方便地录制脚本，通过本地、私有云、公有云和 mTestin 群测，快速、有效地完成测试。

2.6.4 基准测试

基准测试的目的主要是进行与已知系统的比较，包括 App 之前的版本、参照版本、竞品等。

【Testin Tips】
根据基准测试要求，可以通过 iTestin 在本地通过同类脚本的执行，可以有效地判断不同 App 基准测试的结果。

2.6.5 竞争测试

竞争测试是判断 App 竞争使用各种资源（数据纪录，内存等）的情况。

【Testin Tips】
通过 iTestin Suite 进行 App 测试时，所有相关的 CPU、内存、网络等资源数据都会被记录下来，用于竞争测试分析。

2.7 故障转移和恢复测试（Recovery Testing）

通过人工干预手段使系统发生软、硬件异常，通过验证系统异常前后的功能和运行状态，达到检验系统容错，排错和恢复的能力。可确保测试对象能成功完成故障转移，并能从导致意外数据损失或数据完整性破坏的各种硬件、App 或网络故障中恢复。

故障转移测试可确保：对于必须持续运行的系统，一旦发生故障，备用系统就将不失时机地“顶替”发生故障的系统，以避免丢失任何数据或事务。

恢复测试是一种对抗性的测试过程。在这种测试中，将把应用程序或系统置于极端的条件下（或者是模拟的极端条件下），以产生故障（例如设备输入/输出 (I/O) 故障或无效的数据库指针和关键字）。然后调用恢复进程并监测和检查应用程序和系统，核实应用程序或系统和数据已得到了正确的恢复。

【Testin Tips】
在不涉及电源终端或开关机等状态下的故障转移和恢复测试，可以通过 iTestin 对测试 App 及终端在测试过程中的各项参数进行记录，帮助分析、判断测试结果。

2.8 兼容适配性测试（配置测试 Configuration Testing）

兼容适配性测试（配置测试），是核实测试对象在不同的 App、硬件配置中的运行情况，测试系统在各种软硬件配置，不同的参数配置下系统具有的功能和性能。

在大多数环境中，不同终端、屏幕、OS 版本、网络连接的规格都会有所不同，而这些因素都可能运行许多不同的配置环境组合，从而占用不同的资源（如 CPU、内存、浏览器版本、OS 版本等）。

目标：验证全部配置的可操作性、有效性，特别需要对最大配置，最小配置和特殊配置进行测试。

- ✓ 操作系统版本的兼容性：测试 App 在不同操作系统版本下是否能够正确显示与运行；
- ✓ 硬件兼容性：测试与硬件密切相关的 App 产品与其他硬件产品的兼容性，是否可以正确使用；
- ✓ 浏览器兼容性：测试 App 在不同产商的浏览器下是否能够正确显示与运行。

【Testin Tips】
通过 Testin 录制脚本，可以快速提交至 Testin 平台进行 App 的兼容适配性配置测试，测试报告主要包括：

1) UI 截图

2) 运行 Log 日志；
   - PID 跟踪：例如 27665
   - 启动：是否成功，时间，例如 483 ms
   - 错误日志条数 Error log count: 例如 13
   - 警示日志条数 Warning log count: 例如 11
   - 调试日志条数 Debug log count: 例如 4
   - 信息日志条数 Info log count: 例如 6
   - 其他错误日志数 Other Error log count: 例如 36
   - 其他日志自动分析报告
3) 其他辅助信息：
   - 位置和文件夹是否合理；
   - 组件是否正确注册或删除；
   - 评估操作前后，CPU、Memory、Storage 等系统资源的使用

同时，当测试功能点存在交互等复杂操作环节，一个有效的方法是将存在交互的部分、或全部测试任务分解，发布到 mTestin 群测社区，通过分散、专业的测试人员、终端，成本可控地、快速完成兼容适配性配置测试。

2.9 分辨率测试

测试在不同分辨率下，界面是否匹配。

【Testin Tips】
分辨率测试可以上传 App 到 Testin 平台，选取不同分辨率的终端，自动进行。

2.10 网络测试

在网络环境下和其他设备对接，进行系统功能，性能与指标方面的测试，保证设备对接正常。

【Testin Tips】
可以按照网络要求，将测试任务发布到 mTestin 社区，测试者通过符合要求网络的测试终端完成测试任务，并上报测试结果到 mTestin。

2.11 本地化测试

是指为各个地方开发产品的测试，如英文版、中文版等等，包括程序是否能够正常运行，界面是否符合当地习俗，快捷键是否正常起作用等等，特别测试在 A 语言环境下运行 B 语言 App（比如在英文环境下运行中文版 App）是否正常。

【Testin Tips】

可以按照本地化语言要求，将测试任务发布到 mTestin 社区，测试者通过符合要求语言要求的测试终端完成测试任务，并上报测试结果到 mTestin。

2.12 文字测试

测试文字是否拼写正确，是否易懂，不存在二义性，没有语法错误；文字与内容是否由出入等等，包括图片文字。

【Testin Tips】

可以按照文字测试的要求，将测试任务发布到 mTestin 社区，选取合格的测试者分 A/B 组进行测试，测试者通过符合要求素质要求的测试终端完成测试任务，并上报测试结果到 mTestin。

此类发布到 mTestin 的 App 测试，存在明显的“杀虫剂现象”，即由于测试人员的思路不尽相同，每个人测试的侧重点不同，由于都按照测试用例进行测试，但是测试用例一般仅描述系统的一些基本测试项，不会将所有的测试用例方方面面都写到，有时还需要测试人员的经验和素质。所以 A 测试某个产品用了七个工作日，第一天到第四天报出许多 bug，但从第五天开始几乎报不出啥 bug 了。七天后换了 B，B 一下子又测试出一堆 bug，不能说 A 的水平差，只能说该 App 已经对 A 产生了抗药性，这就是测试学中的杀虫剂现象。所以在测试中每次轮流测试最好安排不同的测试人员进行不同模块测试工作，以避免杀虫剂现象产生。

2.13 发布测试

主要在 App 发布前对说明书、广告稿等进行测试。

【Testin Tips】

发布测试可以由 App 产品、客服团队内部测试，或发布测试任务到 mTestin 进行测试。

2.13.1 说明书测试

主要为语言检查、功能检查、图片检查。

- ✓ 语言检查：检查说明书语言是否正确，用词是否易于理解；
- ✓ 功能检查：功能是否描述完全，或者描述了并没有的功能等；
- ✓ 图片检查：检查图片是否正确。

2.13.2 宣传材料测试

主要测试产品中的附带的宣传材料中的语言，描述功能、图片等。

2.13.3 帮助文件测试

帮助文件是否正确，易懂，是否人性化

2.14 回归测试

回归测试是以上所有测试完成后的一个最为重要的环节，是 App 发布、维护阶段，对缺陷进行修复后的测试。

目的是验证缺陷已经得到修复，检测是否引入新的缺陷；

流程：

1）在测试策略制定阶段，制定回归测试策略；

2）确定需要回归测试的版本；

3）测试版本发布后，按照回归测试策略来执行回归测试；

4）回归测试通过，关闭缺陷跟踪单；

5）回归测试不通过，缺陷跟踪单返回给开发人员，开发人员重新修改 BUG。再次提交给测试人员回归测试

测试策略：

1）完全重复测试：重新执行前期设计的用例，来确认问题修改的真确性和修改的扩散局部影响性；

2）选择性重复测试：

a) 覆盖修改法：针对被修改的部分，选取或重新构造测试用例验证没有错误再次发生的选择方法；

b) 周边影响法：该方法包括覆盖修改法，还要分析修改后对扩散的影响；

c) 指标达成法：先确定一个达成的指标，基于这种要求选择一个最小的测试用例集合。

【Testin Tips】

由于回归测试是各项系统测试的重复，所以通过 Testin 所提供的各种测试工具与方法仍然是适合的，之前测试录制的脚本可以继续使用。根据回归测试的终端、可以分解任务执行。