Software Engineering Intro.

Final Project Paper

wacky6 / Jiewei Qian

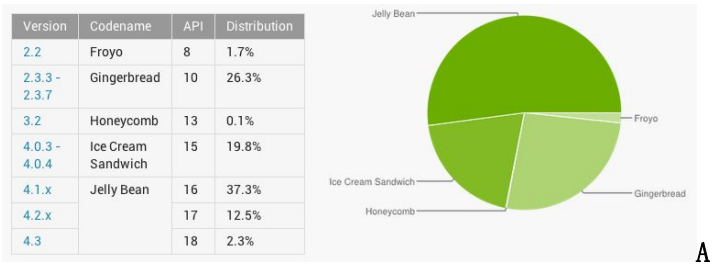
This is the final project paper. You should not copy it, because information in it is out-dated.

If you want to include this in your research, please provide a link to this document. (Completed on Dec 2013.)

: (

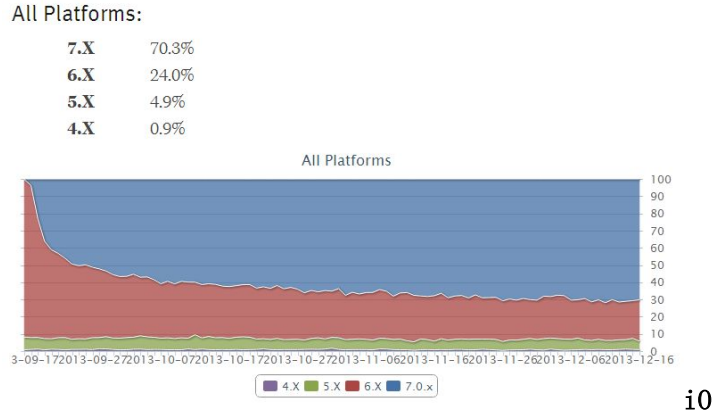
平台碎片化对软件开发的影响

综述：Android 从诞生开始就受到热捧，并日益普及。它有别于以往的手机及移动终端操作系统，其独具的开源性、系统廉价性和提供给第三方大自由度的创新空间，以及不受硬件约束的优势，获得了广大开放社群的支持。然而随着 Android 平台的深入发展，系统版本以及硬件的碎片化问题日益严重，使得该平台上应用开发的复杂度与成本愈来愈高，已经开始明显影响到开发者。



Version	Codename	API	Distribution
2.2	Froyo	8	1.7%
2.3.3 - 2.3.7	Gingerbread	10	26.3%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	19.8%
4.1.x	Jelly Bean	16	37.3%
4.2.x		17	12.5%
4.3		18	2.3%

Android 平台各版本分布 （2013 年 11 月）[1]



iOS 平台各版本分布 （2013 年 9 月-2013 年 12 月）[2]

现状

当前，版本碎片化是移动开发中不可避免的问题。左侧数据是近期 Android 与 iOS 两大平台的不同版本分布状况。

从以上数据可以看出，Android 的版本分布较 iOS 而言更加分散，有很大一部分用户仍在使用老版本的 2.3 内核。11 月底，使用最新版本 4.1+内核的用户仅仅略多于 50%。

而 iOS 方面，自 9 月 iOS7 发布以来，大多数用户已经选择了升级（从左侧折线的变化趋势来说，用户还是很快的就选择了系统升级），12 月 iOS 7 的比例已经达到 70%。这意味着开发者在开发 iOS 平台应用时，可以大胆地采用更新版本的 API，而不必为了兼顾旧平台而浪费开发精力。即便需要照顾尚未升级的用户，也可以使用 12 年发布的 iOS6 API。而对于 Android 而言，现在开发的应用（包括应用升级），如果要保证绝大多数用户多可以使用，仍然只能使用 2010 年发布的 Android 2.3 SDK。这导致 Android 新内核的功能很难在短时间推广，开发者也更难以较低的开发成本为用户提供更好的体验，也一定程度限制了开发者的创造能力（老版本 API 的功能比较少，一些功能的开发必须要使用新版本 API）。同时，一些使用旧 API 的程序在更新的系统上会出现一定的兼容性问题，导致用户体验下降。平台碎片化引起的移植性与通用性问题是软件开发的隐性成本之一。

原因分析

1. **Android 开源开放，可以支持的硬件平台多。容易移植到不同硬件上。**

生产不与单一的设备制造商有关。[3]

目前 Android 平台主要有 Qualcomm、NVIDIA、Samsung 三家厂商生产 SoC 芯片。

各芯片存在各自不同的扩展指令和特性，开发者需要针对不同设备优化。

2. **Android 硬件生产的准入要求低。小厂商易投入生产，但后期跟进能力弱。**

很多小厂商也能根据 Android 设备的要求自主设计，以较低的成本生产硬件设备。

低廉的价格吸引大量的使用者。而小厂商与大厂商相比，推送系统更新的能力相对弱。

3. **用户层次多而杂，系统更新推送的效果不佳**

Android 设备分布于 400 元至 7000 元的价格层次，能够满足不同层次消费者的需求，不同消费者对待设备系统的态度不同，部分消费者需要手机仅仅是使用社交应用，对新功能、新特性并不关心，这导致用户不愿主动更新系统。而 iOS 面向追求潮流的年轻人，有明确的用户群，大多用户愿意体验、尝试新功能，主动进行系统升级。这导致 Android 新版本需要很长的时间才能被用户接受。

4. **Android 用户基数大。小百分比也意味着大数目。**

开发者不能轻易放弃小比例的老版本用户推行基于更新版本 API 的程序。

与 iOS 的单架构相比，Android 平台的开发（尤其是游戏开发）需要考虑真对各个不同硬件平台的优化。而 iOS 平台开发，仅仅需要使用苹果提供的公用接口，便可以发布通用安装包。

Android 应用需要对不同架构发布不同的数据包以提升应用性能、提升用户体验。这给开发者造成了额外的负担（不仅仅需要学习不同架构的开发文档，还要在主程序中添加一个抽象层以便上层结构的开发）。这样的负担对与中小型开发者特别明显。中小型开发者不具备大型企业的资源，不能像大型企业一样分开若干组分别处理各个平台、架构的抽象层代码。但小企业以及初创企业更能够开发出具有革新性的 App。[4]在最近一次关于新发布 App 的调查中，新应用 70%以上均出自 iOS 平台。

另一份报告指出：“iOS 每名使用者替开发人员带来的营收仍是 Android 的 5 倍。同样的 App，在 iOS 平台上能赚 1 美元，而在 Android 平台上只能赚 0.24 美元。开发者们发现为苹果设备开发软件比为安卓系统设备开发要来得容易，并且可能更赚钱。而开发一个 Android 应用不但赚钱少，还要面对碎片化问题，开发者当然会优先选择 iOS 平台了。”[5]这指出了 Android 碎片化带来的问题，现在移动设备行业最核心的是 App 开发，Android 的碎片化导致大量中小型开发者（尤其是很多初创企业）放弃 Android 平台，这会导致 Android 平台在未来失去可观数量的优质应用，从而导致用户流失。

可能的解决方案及分析

1. **使用可以跨平台编译的开发环境和开发包**

优点：需要阅读的参考资料少，对一般应用来说不需要特别考虑移植性问题。

好的 SDK 能够做到多个平台上生成的最终程序有一致的界面与使用体验

适合小型开发者快速开发（RAD）且成本低

缺点：不能使用平台特有的特性，功能上有限制

部分 SDK 需要单独购买商业授权或不允许进行商业开发

此类 SDK 较少，多为开源包，支持服务缺乏，更新缓慢

样例：Qt 开发包

2. **自行编写抽象层，分离不可直接移植的代码，使用自顶上下的开发方式**

优点：能够较好的使用平台的功能，执行效率较跨平台开发包而言更高

不需要额外授权，对抽象层进行维护的开销可控

能够针对最主要平台提供最优化的程序

分离代码的结构能以较低成本提供次要平台的可移植性

能够提供一定的基于平台的功能

对大公司而言，可以降低后继应用的开发成本，共享基础框架

缺点：前期开发需要有一定的投入

对已有项目进行修改的成本较高，容易引进 BUG

仍需要考虑不同平台使用不同语言导致的问题

样例：linux 内核开发 asm/、arch/存放有关特定平台的 linux 基础功能实现

腾讯手机 QQ 客户端

3. **分别开发不同平台的代码**

优点：能够对不同平台提供最优的代码

对游戏开发商而言，可以以可接受的成本最大限度利用处理器性能

缺点：平台数量大时，开发成本会明显提高

需要更多人力投入开发，开发人员精力分散，降低效率

应用维护成本高

样例：Gameloft 对 iOS 与 Android 分别开发游戏包

4. **利用 HTML5 等标准化的新技术基于浏览器开发应用**

有点：学习成本低

具有最好的兼容性（只要客户端浏览器支持 HTML5）

可以通过 HTTP 请求的相关信息真对不同屏幕、系统的终端设备优化

容易修复漏洞（只需更改服务器代码，不需客户端有操作）

缺点：功能受限制

运行效率较低（需要处理 HTTP 代码而不是字节码或者机器码）

不能支持较老的终端

用户体验不可控（受浏览器影响）

总结

软件和软件运行的平台是不能分离的，优质的平台可以降低软件开发、维护的成本，好的软件为平台吸引更多的用户。软件的目的是实现用户需求的功能，提供良好的用户体验。在平台多元化，设备多样化的今天，软件在不同环境下表现差异的问题是需要妥善解决的。软件厂商应意识到这一现状可能引起的诸多问题，并妥善解决，平台提供与开发商也应该在平台设计和运行时注意到碎片化的问题，并且加以防范。只有形成软件与平台相互促进的良好关系才能保证软硬件生态系统的正常运行。

参考：

[1]. Android 平台数据来源：Android Developer

http://developer.android.com/about/dashboards/index.html

[2]. iOS 平台数据来源：

http://david-smith.org/iosversionstats/

[3]. 引述自 IBTimes/Lisa Mahapatra

[4]. 引述自 Paul Graham, Hackers & Painters （保罗•格雷厄姆，《黑客与画家》）

[5]. 引用自 IBTimes

http://au.ibtimes.com/articles/526287/20131130/google-android-apple-ios-vs.htm