NISTIR 7838

Initiating Mobile Software Development - Lessons Learned From A 12 Month Project

Frederic de Vaulx
National Institute of Standards and Technology
Information Technology Laboratory

Paul Khouri Saba
National Institute of Standards and Technology
Information Technology Laboratory

Marcus Newrock
National Institute of Standards and Technology
Information Technology Laboratory

Bertrand Stivalet
National Institute of Standards and Technology
Information Technology Laboratory

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7838

Initiating Mobile Software Development - Lessons Learned From A 12 Month Project

Frederic de Vaulx
National Institute of Standards and Technology
Information Technology Laboratory

Paul Khouri Saba
National Institute of Standards and Technology
Information Technology Laboratory

Marcus Newrock
National Institute of Standards and Technology
Information Technology Laboratory

Bertrand Stivalet
National Institute of Standards and Technology
Information Technology Laboratory

December 2011

U.S. Department of Commerce
John Bryson, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher,
Under Secretary of Commerce for Standards and Technology

Initiating Mobile Software Development - Lessons Learned From A 12 Month Project

As mobile devices' capabilities expand, today's mobile software ecosystem evolves at an unprecedented speed. This evolution makes mobile software development both more common and a critical part of the development cycle. Moreover, the ever-increasing number of new development platforms can make it difficult for developers new to mobile application development to stay current with the latest technologies. Are there elements developers can leverage to accelerate the learning process of mobile applications development? In this article we explore what we believe to be a set of key elements critical to building our applications and optimizing our development process. These elements are based on our experience carrying out a 12-month development cycle resulting in three prototypes for both tablets and smartphones.

The need for a prototype using regular mobile devices - Smartphones and Tablets

The project started when we were tasked with providing a group within our laboratory with software development expertise. This group, specialized in usability, was writing usability specifications for law enforcement sponsored projects. One major component of these projects was to make use of mobile devices to facilitate the work and communication of law enforcement officers while leveraging existing COTS (Commercially Off-The-Shelf) mobile devices. Some of the technical reasons why the use of such devices was appropriate were:

- The commercially produced mobile devices showcase the state-of-the-art in terms of technologies readily available in mobile devices realm,
- These devices already come with a wide range of communication protocols like Wi-Fi or 3G.

Our first objective was to produce prototype applications using state-of-the-art technologies of mobile devices like touch screen capabilities and portable accessories. These prototypes would implement usability-centric specifications with most of the front-end capabilities and a simulated back-end. Later in the project, a second objective was to implement the back-end of the prototypes.

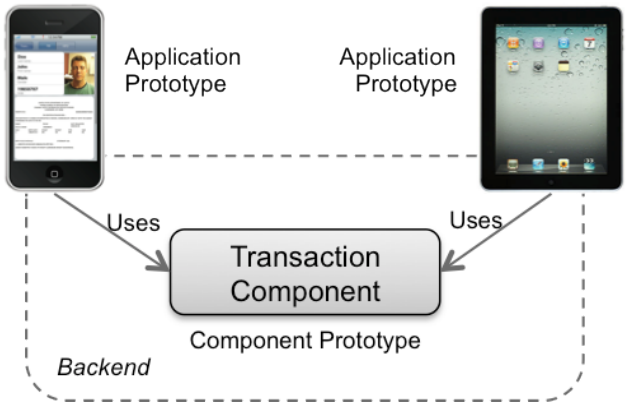

Figure 1: Project Overview

Figure 1 represents the combined view of the three prototypes we developed and their relationships. There are two application prototypes and one component prototype. The first application prototype is a tablet-based application whose purpose is to reproduce a process followed by examiners when analyzing latent prints from a crime scene. The second application prototype is a smartphone application with a fingerprint reader attachment, whose purpose is to help law enforcement officers in the field get fast identity

information. After the completion of the first stage of the two application prototypes, we started working on the component prototype, which would be shared by the two application prototypes. The purpose of the component prototype was to replace the application prototypes' simulated back-end with an implementation able to handle transactions following a biometric transmission specification standard, which defines the content, format, and units of measurement for the exchange of biometric information.
The different elements we emphasize in the next sections apply to both prototypes and commercial applications. These elements represent lessons learned during our development cycle starting with minimal to no experience in mobile software development but with a strong knowledge of Object-Oriented software development.

Mobile development platform, specifications, patterns, fundamental concepts and tutorials

Dealing with mobile devices
It is important to be aware of the different aspects of the mobile platform. It is not just the use of a specialized SDK (Software Development Kit) for a given language. It also involves the use of this SDK on a given device that integrates a set of specialized hardware internal or external like wireless connections, camera, or external fingerprint reader running on a constrained mobile environment. Some platforms tightly integrate both the SDK and device (e.g., iOS and Apple devices), others allow for more decoupling between hardware and software (e.g., Android and Samsung or HTC devices).

Understand the fundamental concepts of the SDK
Each platform's SDK has its own way to implement design or architecture patterns needed to create a good solid application. Furthermore the mobile SDK is often composed of libraries optimized to run in a constrained environment. The number of libraries within the mobile SDK is also usually less, compared to a regular desktop SDK. It is important to understand how the mobile SDK creators interpreted the different patterns (i.e., design or architecture) and how they intended the developers to use them[1] in order to create applications. This can be translated into understanding what patterns are used at the presentation layer to implement window management, events handling or notifications management in order to make the best use of the SDK.

Understand the fundamental concepts of the language
Although a great focus is directed to the mobile SDK itself, the mobile SDK draws major implementation practices from the programming language it is based upon. Understanding its basic concepts is self-evident but need to be noted as one doesn't want to create unnecessary or inefficient code on a constrained platform.

[1] e.g. in object-oriented languages the delegation design pattern delegates responsibilities from one object to an other and can be implemented to allow for behavior customization without the use of subclasses.

Have well defined GUIs (Graphical User Interface) and workflows

Mobile applications are by definition mobile

A lot of the critical features of a mobile application are used while in motion when the full attention of the user is not on the application itself. Applications like GPS, email, web searches, specialized web sites are great tools coming in support of activities the user is performing. Mobile users don't have the time or focus to handle unnecessary functionalities of tasks the mobile application could provide.

This means that the application's workflow and layout must be well designed to ensure its primary function is carried out with maximum efficiency.

Workflow design

Displaying content to a user on a mobile device differs from traditional information displays. All information presented has to be the minimal required. Navigation between pages also has to be intuitive and mapped out by the developer prior to the implementation. When designing workflows, user interface design guidelines and best practice documents should be followed. Whenever possible, views and frames templates extracted from user stories and scenarios should be used to remain consistent between page views.

GUI design

Individual classes of mobile devices should be considered when managing the screen "real estate". Not all mobile devices' displays provide the same amount of space for an application to display output. Thus, presenting the most efficient data and controls for a given task might become a challenge if for instance one started to develop an application for a tablet and then decided to port it to a smartphone of a smaller size without using resizable graphical widgets.

The ways a user interacts with the device should be one of the first considerations when designing the user interface. The user of the application cannot get lost in the flood of information. The display must be clear, concise and easy to use. Because of its small size, it has to display only minimal and relevant information on the screen. The mobile application should be very intuitive.

It should happen early in the development

Once the major use cases and scenarios have been translated into concise pages with a consistent look and feel and with a fluid work flow, it is a lot easier to focus on the technical parts of the mobile platform of choice. Defining this before implementation begins will ensure that critical user components are not forgotten when trying to understand the SDK framework.

Seek advice from usability experts

In our case, the usability group provided us with well-defined scenarios and page layouts. We started from there to define more accurate processes and page flows. This allowed the team to locate more easily in the SDK what libraries or components to use and make sure that the chosen platform could handle what needed to be done. When access to usability experts is not possible, papers like the "Three Layers Design Guideline for Mobile Application" [1] can be a good start.

Develop within the higher layers of the SDK

Higher layers are more application driven
A lot of SDKs are designed in layers to facilitate ease of use. The higher levels of the stack present APIs whose sole purpose is to allow developers to worry less about solution and implementation issues related to the hardware and focus more on the resolution of the application scenarios. It also allows the SDK developers to control parts of the system while still giving the application developers the possibility to tweak them (i.e., the implementation of error management in APIs to access documents).
By designing content with components from the higher layer of the SDK, the developer can focus on what content is given to the user rather than how to present it. The developer only has to consider what template best fits the information being conveyed to the user.

Easier communication with the hardware stack
Staying within the higher layers of the SDK allows easier access to the hardware stack, as most of the implementation issues have been dealt with. For instance, the developer can use special APIs to access sensor information that is available in the hardware. It is a good practice to first check if a solution can be implemented using an API from the higher layers and then delve deeper into the layers of the SDK if more control is required.

Limit the capabilities of the defined APIs
As layers of the SDK get stacked on, APIs get less flexible as they become more driven by application functions (i.e., temperature average) and less by hardware features (i.e., instant temperature sensor). This means that there needs to be a balance between the ease of use of a given feature of the SDK and the level of control the developer requires.

One of the biggest benefits of developing within the higher layer of the SDK was when we had to develop a touch-screen equivalent of an examiner's loupe on a crime scene latent print. We were unsure of how to approach this, and it turned into a very complex interaction of managing layer geometry, examiner standards, and touch. In what is a very simple metaphor, an examiner looks through a magnifier (loupe) at a crime scene latent.

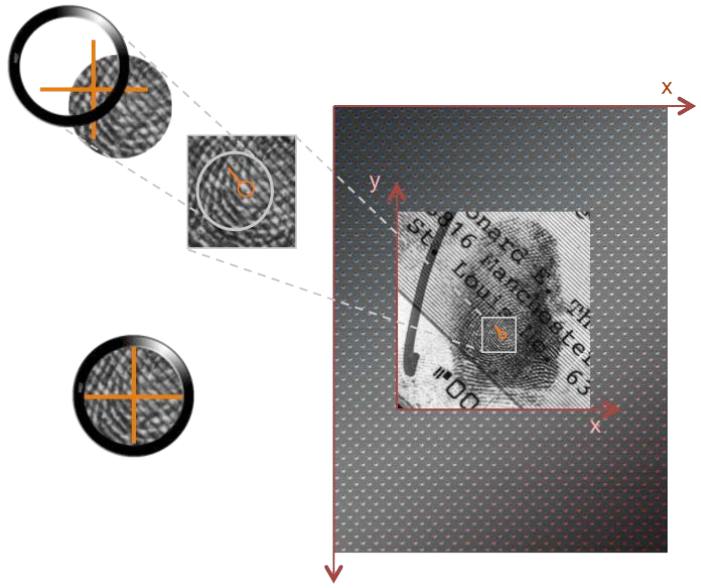

Figure 2: Loupe Layer Decomposition

Where the SDK helped was the management of views and sub-views. With touch-based control, it was helpful that the coordinate of a touch point could be checked in any of the views. This helped the translation of one coordinate space, the frame, to another, the location of a fingerprint minutia marker (a fingerprint's major feature) on a latent print.

Develop within the lower layers of the SDK

Performance and low-level needs

Unlike higher layers of the SDK, the lower layers' target area is concentrated around performance, low level needs and greater control of the SDK like the manipulation of file access or memory. When it comes to accessing files within the device, whether it's a smartphone or tablet, developers must be aware of the complexity that they might encounter; because at this stage file access at lower layers of the SDK has a different implementation than at the higher levels. For instance, implementing file access at lower layers would be using C code and more specifically the old C function named "fopen". On the other hand, the implementation of file access at higher layers in Objective-C is a simple function from the SDK that uses a File Manager with the implementing language. As for memory manipulation, developers should be careful when allocating/deallocating memory with C as well as retrieving the size of memory types as using those techniques incorrectly may lead to a failure (i.e., segmentation fault).

Getting close to the device architecture

One experience we had using C code was importing WSQ (Wavelet Scalar Quantization) encoding libraries for use in the smartphone project. Because C was supposed to be usable in Objective-C code, we were expecting there to be very few changes required. When we began to build the library for the smartphone, we did encounter some unexpected difficulties. Attempting to implement file access through "fopen" caused a compile-time error, so how the file was loaded into memory and passed to the WSQ encoding methods had to be changed to use Objective-C. Another issue we had was with the results of the actual encoding. The byte ordering was incorrect, and although tests for endianness[2] and documentation

indicated that the device was little-endian, the results on the iPhone device did not match the results in the smartphone simulator. We found the solution to this was to add a little-endian compile flag to the WSQ library build.

Development through reusable libraries

The need for code reuse

As we were finishing the work on the two application's front-ends, we started the work on the component prototype. We realized that there were functionalities being developed for one application that had potential use in the other application. Both applications had to communicate in a way that would be valid to the law enforcement mailing systems. We had to design a module that could model and generate all types of biometric transactions (e.g., fingerprint, facial, palmprint, iris) for a given standard. We also needed to ensure that we could anticipate the need for the change of some of the implementations like from email to web services. Therefore, we decided to integrate and reuse common modules in two different applications in order to gain development time and flexibility.

Specialized libraries are easier to manage

Figure 3 describes the applications back-end module decomposed in specialized libraries (i.e., transaction process, encryption capability, image encoding capability or message formatting capability)

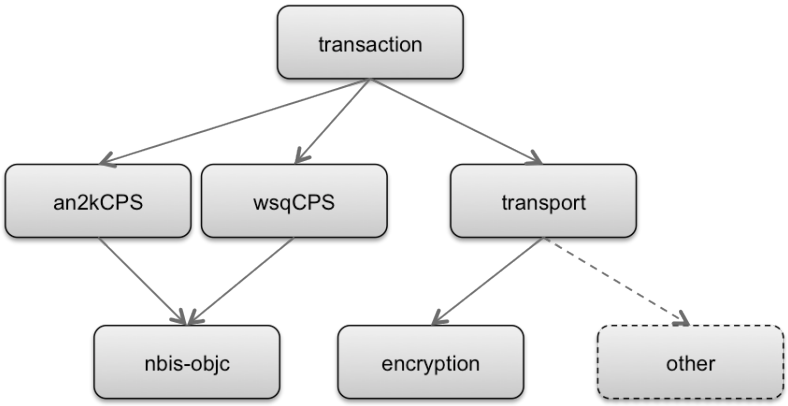

Figure 3: Dependency Graph

In addition to allowing for better code re-use, specialized libraries create an integrated structure only coupled when necessary. It makes it easier to control the scope of what is being developed and to define clear public interfaces. It also makes it easier to isolate low-level code that is close to the device architecture and thus shields the rest of the application more efficiently. Specialized code that deals with a very concrete set of features is also a lot easier to manipulate and test.

The architecture described in Figure 3 allowed us to separate the different technical challenges so they could be easier to deal with. For instance we put all the code that came from an external set of libraries dealing with biometric data in the nbis-objc library. It was first compiled with a desktop OS SDK and had to be recompiled for the device SDK. We then abstracted out some APIs as described in the "*Develop within the higher layers of the SDK*" section in order to make it easier to use at a higher layer and also to specialize their functionalities (i.e. an2kCPS for message formatting and wsqCPS for image encoding).

The Definite need for testing

Quality Assurance
Software quality assurance (QA) is a very important step that developers should take into account when they are developing mobile applications. It helps verify that the product has been implemented correctly in order to meet the specified requirements and as D. Franke and C. Weise mentioned in their article [4]
"Another reason is the diversity of platforms that are available on the mobile market (Android, iOS, Windows Mobile, Symbian, ...). Each platform has its own architecture and design concepts."
In addition, each platform doesn't necessarily have the same relationship with the underlying hardware (i.e. 3G capability unavailable or differences in driver implementation). And to add to the complexity, most of the development platforms provide an emulated environment to test the mobile application without the use of the hardware. It is very helpful during development but adds an additional architecture to the mix.
This results in the necessity for having a good test plan in order to make sure that all required functionalities are implemented.

Development infrastructure involving several architectures
As mentioned in "Quality Assurance", developing a mobile application will generally result in using at least two different architectures like the simulator (e.g., i386) and the device (e.g., ARM). This will involve different build schemes to compile the code against a given architecture and include the compiled code in a testing plan that will also involve different architectures.
In the case of the statics libraries described in the "Development through reusable libraries" section, three targets were created within the project for:

- The development of the library.
- The development of the "logical" unit tests.
- The development of the "application" unit tests.

"Logical" tests were conducted to directly test the library (context library) using the simulator and "application" tests were conducted having the library (context application) used in an application destined to be pushed on the device. The later would be tested on both the simulator and the device to make sure we weren't using forbidden or undefined artifacts. Figure 4 summarizes this testing scheme.

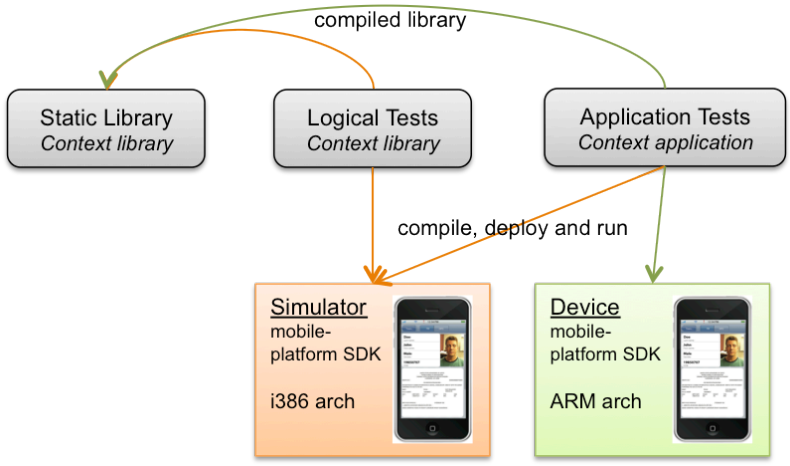

Figure 4: Testing Plan

The testing plan also has to take into account the fact that it is not always possible to test the entire application on the simulator. The main reason is that not all hardware related libraries are implemented or

perfectly emulated on the simulator. In our case we had an external attachment and its library could only be used when using the actual smartphone with an attachment. The attachment also prevented direct test monitoring so debugging had to occur through prompts and analyzing crash logs.

The Need for a well-defined development and build scheme

Modules development decomposed in specialized projects
As we started working on the different libraries, we realized that it would be a lot easier to manage each library as a separate project. Having one project space per specialized library allowed us to make sure that everything needed to develop, test and run against a single library was bundled together. It also made it a lot easier to share the projects among developers when needed.

Put together a consistent configuration within projects
Each project needs to be configured so it follows a consistent template for the project itself as well as the integration hooks to other projects. Each of our specialized projects was generating shared static libraries. As explained in "The Definite need for tests" section, each project had a consistent development scheme based on targets for efficient testing.
The use of external libraries (i.e., coming from another specialized project) within a specialized project was facilitated by the setup of a build process. Shared files like class headers or interfaces were stored in a unique location and the generated binaries/libraries were also stored at a unique location. This created a systemic way of handling external dependencies and allowed us to focus more on the current library development. Our build path, shared files paths were always pointing to the same folder using relative paths, which subsequently made it easier to share the projects among team members.

Automate builds to ensure updated libraries during development process

The need for build scripts
As some of the projects rely on other projects' generated libraries, we need to make sure that we always use the most up to date libraries. One way to guarantee this is to rebuild and test the project thus generating the library each time. In the mobile software environment, to generate a library can actually mean to generate a library per platform (simulator and device architectures). Therefore it makes sense to produce build scripts to automate the generation of the libraries.
As mentioned in "The Need for a well-defined development and build scheme" section, we generated all libraries in a unique location. Having made the choice to work with independent modules, this solution allowed us to centralize the management of the project operation. After each modification, a script would rebuild all libraries, and then would test if the change fits into the overall project. This allowed us to continually use the latest version of the libraries developed, and avoid version conflicts.

Memory management

Understand the mobile platform memory management concepts
Although mobile devices' hardware gets more powerful with increased CPU speed, memory size and specialized chips, we are still dealing with embedded systems whose memory needs to be managed carefully. Each SDK for mobile development provides different tools to manage memory. Studying how the SDK manages memory and what the best practices are when using the language is a critical step to avoid producing mobile applications that crash. For instance, at the time this paper was written, platforms

like Android provided garbage collection tools whereas iOS did not. This made it important to understand the basic memory management rules before starting the development process.
Garbage collection alone is not enough, it is still important to understand the concepts used to manage some elements like how images are allocated in memory, or how the different memory layers work and what types of objects they keep.

Automatic Code Analysis
Moreover, IDEs for Integrated Development Environments (e.g., Xcode) became more and more aware of memory issues in such a way that they have been providing powerful tools (i.e., Automatic Code Analysis) that give developers the ability to detect potential leaks or allocations before even running their program.

Tracking Down Performance - memory leaks
As mobile applications are intended to run in a constrained environment with limited memory, it is important to pay special attention to memory leaks. As explained above, it is important to first have a good understanding of memory management concepts related to the platform. In addition, the development environment usually provides efficient tools to track down possible memory leaks (i.e., the Eclipse Memory Analyzer used or Instruments). Such tools can be used to generate memory dumps that can be analyzed for leaks. In fact, it has been a great help to us as developers to detect memory leak and/or memory allocation issues that came up in our prototypes especially when we had to deal with a large amount of images.

Conclusion

We were able to draw upon several lessons from this work. We feel that the key elements described in this paper are a good starting point for developers new to mobile software development and can help speed up their learning process. Some of the key elements to keep in mind are GUI design, platform environment, test plan, and memory management.

Beginning mobile development can be a daunting task. There are a lot of questions that need to be asked, and there are a diverse number of environments to develop code in. Decisions will range from what language to implement your project in, to the best, most minimal way to display information to the user. When you have a chance, look at your favorite mobile app and consider the details that went into the design and implementation.

References

[1] Ayob, N. and Hussin, A.R.C. and Dahlan, H.M.. Three Layers Design Guideline for Mobile Application

[2] Endianness Wikipedia
http://en.wikipedia.org/wiki/Endianness

[3] Dominik Franke and Carsten Weise. Providing a Software Quality Framework for Testing of Mobile Applications

Disclaimer: We identify certain software products in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available for the purpose.