The Dirty Little Secret of iPhone Development

Adrian Kosmaczewski

2008-12-23

This is happening right now, at a web agency near you.

The dot-com boom of the 90's spawned a brand new generation of coders and software developers, including me, by the way. While before that time the term of "software developer" might have been reserved to system programmers fluent in C, COBOL, C++ or other languages, right now the vast majority of developers I know spend their time writing web applications, either public or in a private intranet, in J2EE, ASP.NET, Rails, PHP, you name it.

I have said before that writing web applications should be taken as seriously as writing desktop systems. Call me names if you want, but I'm a big fan of Joel's Test.

However, after all this years, after the Chaos reports, after Peopleware, after the Mythical Man Month, people still treat quality as an afterthought. And also complain about how much software sucks, how expensive it is, and how late it arrives, by the way. Now that the iPhone SDK is widely available, that the App Store is selling more apps that we could have had imagined 6 months ago, many web agencies want to jump to native iPhone development contracts, which are hype and nice and pricey and whatnot. Which is only going to make things worse.

The dirty little secret in this story is this: **iPhone development looks more like developing applications for a desktop operating system, and less, much less than web development.** And I'm frightened to see some small shops (and even bigger ones), who never attained a real level of professionalism or quality in their software tasks, starting projects and realizing, later, when they are over budget and behind schedule, that this kind of applications requires a different mindset.

Let's repeat something that you should know by now: web apps are easy to maintain. To begin with, you just have one instance running of it, and as Paul Graham said, you can write them in any language you want, you can release bug fixes with your application running, monitor performance issues live, change its appearance quite easily and for everyone at once, etc. This is probably the single biggest advantage of web apps. With AJAX we even got the possibility

of going beyond in terms of user interface, responsiveness, perceived speed, you name it, and today the number of famous web apps is outstanding.

However, web apps run into this terrific (and terrifying) sandbox called the browser. They (normally) cannot access your file system, they cannot open other applications and interact with them - well, with some custom URL schemes like lastfm: mailto: or skype: you might have the ability to do some things, but certainly not much more than activating the application in some limited way as a help for the user. You cannot access the user's hardware directly - well, again, you can access the webcam through Flash, or you can trigger the window.print() method to have the native print dialog pop up, but that's more or less the maximum you can do.

In the iPhone, there is a similar situation. You choose to create a native iPhone application over a web one when you require one or many of these things in your application:

- GPS geographical data;
- Accelerometer information;
- Photo camera or library;
- 3D graphics;
- Complex animations;
- Address Book entries;
- Sound recording and playback;
- etc…!

I've talked about the dichotomy between iPhone native vs. web apps in my speech during the iPhone conference where I used this graphic, which might help those having to decide whether they should do a native or a web application:

The tradeoff, basically, consists in knowing that having the ability to access these features, means that you are giving up your capacity to roll out quick upgrades to your code. You have to depend on Apple's own review process timings for that, which, as far as I've seen so far, cannot be predicted. And finally, you depend on the user's final will to update your application!

Similarly, you also lose the ability to create these applications using your common, well-known, garbage-collected programming language, but you'll have to deal with Objective-C exclusively, together with its weird syntax, possible memory leaks, eventual out-of-memory notifications, lazy loading techniques, compiler warnings and errors, and seeing the MVC design pattern even in your wildest dreams at night. Regarding the syntax, I admit that I love it, but it took me a while to get used to it, when I started playing with it back in 2003.

Given these conditions, when you start an iPhone project you will have (let me be very clear about this, so I'll repeat) you will have to dust out your good old desktop application programming techniques (or learn if you don't have them), read Code Complete again (which I'm doing right now) and take all the advice that you can from C and C++ programmers who have been working in this

kind of stuff for the past 20 years.

I've even done my Master's degree project in C++ because of this: the mindset required for iPhone apps is not the same as for your day-to-day web application, and is closer to that of a desktop application. I've been doing web apps for 12 years now, and desktop apps for 5 years (mostly in C#, Objective-C and C++); that's my ground for stating this. You require a spec, you require tests, you require testers, you require daily builds, you require bug databases, you require quiet working conditions.

You require at least an 11 in Joel's Test to know that you're doing fine, but not only that: you need to know about pointers, you need to know C, you need to know that your code runs in an fragile environment with EXC_BAD ACCESS (SIGBUS) and low-memory warnings and stuff like that happening when you expect it the least.

Unfortunately, from what I've seen so far (at least here in the Lake Geneva region) many companies are spending much more than they intended to for rolling out their iPhone apps; these are real quotes from people I've dealt with in the past 6 months:

- "Oh, we do not write specs, we prefer to modify the code as we have ideas!";
- "Oh, we do not write tests, we do not have the time for that!";
- (in general) "Oh, we do not work like that here. We do what the client asks and that's all.";
- "We will not follow Apple's iPhone Human Design Guidelines at all; we have our own. Users must double-click on buttons, so they feel more comfortable while using our application";
- "We want our [PUT YOUR FAVORITE UIKIT CLASS NAME HERE] to have [SOME IMPOSSIBLE FEATURE WHICH DEFIES GRAVITY]. Easy, right?"
- "We have to remove this XYZ feature, right now!", which means rewriting half of the code and leads to this: "What? So much? For such a small change?"
- "Why have you spent all that time 'fixing memory leaks'? I won't pay for that. Please concentrate in the new features we've asked. By the way, what are memory leaks?"
- "Here's the styles document you asked, with the colors and fonts for the application" and the guy provides me with… a CSS stylesheet. Which references a font not available on the iPhone, by the way, and with colors in hex format.
- "We want an animation at startup, like the Chanel application" (everyone wants to release the next Chanel iPhone app these days) "here's an [animated GIF / Flash movie / PowerPoint slides] with the animation for you."
- "Our team has slightly modified the code when you weren't there, but it does not work any more, could you fix it please?"

And of course, the all-time winners: the support tickets stating that "everything is slow" without further indication, or that "the application hangs", without any details in it.

OK, I confess, I'm a bit of an extremist here; many of the quotes above are valid in some contexts. But not those of the small-to-medium sized iPhone projects I've been involved in lately, with the urgency of releasing the code as fast as possible, just for the sake of being there, in the App Store, right now.

There's a long road ahead. The problem, again, is not the technology itself, but the people involved in these projects (me, for example :). There's a bit of what Joel mentioned a while ago, about the perils of Java schools, which actually only has to do with developers, but there's also the issue about teaching the clients the limits of the platform, and about creating a strong software engineering body of knowledge in companies which usually did not need it previously.

We've been doing web apps for so long that we've forgotten how to sit down, take a deep breath, fix that goddamn memory leak, and realize that, as Brooks and Joel said, good software takes time.