

Epic Code Re-use

Brent Simmons

NewsGator

@brentsimmons

inessential.com ranchero.com

taplynx.com netnewswireapp.com

I'll tweet/blog a download link

once I get this uploaded

How to Make a Huge Nightmare-y Mess

- Start your app in 2002
- Revise architecture in 2004
- Wait for iPhone/iPad to be invented
- Don't share code (for some good-ish reason)
- End up with five different codebases

How to Fix the Mess

- Duh. One codebase for all apps.

Elements of Code Re-use

- Repository for all shared code
- Cross-platform-y-ness
- Architecture

Shared Code Repository

- Single repository is less complex
- Mercurial sub-repos are cool (git, svn, etc. may also be cool too)
- Create as a sub-folder in each project

Share code project

- Shared code doesn't have to have its own project — but it helps
- Unit tests
- Custom test apps
- Easier to manage/view/organize: you can stay in Xcode

How to include shared code

- Just include the files
- Or make a static library
- Or make a framework
- If library/framework — use Xcode project dependency
- You can still debug

Hosting services

- I like Bitbucket
- I like GitHub

Candidates for shared code

- XML, JSON webservicess stuff
- Data model
- Foundation category methods
- Keychain
- Utilities
- View controllers and views, sometimes
- Everything else

Cross-platform code

- Three platforms, sort of (Mac, iPad, iPhone)
- Two platforms (Cocoa, Cocoa Touch)

Foundation

- On all three platforms
- Other frameworks exist on all three

AppKit/UIKit replacements

- Not really replacements
- CGImageRef
- CoreText
- CoreAnimation — CALayer is groovy
- Quartz drawing

HTML/CSS/JavaScript

- You can do amazing UI — see WWDC videos on things like CSS transitions
- WebView/UIWebView are similar

Architecture is golden

- Modular
- Self-contained (few or no dependencies)
- Simple APIs
- Pretend you're Apple, and you're making this for other developers
- Loose loose loose loose loose
- But it's okay if it's kinda tight the first time you do it — you have to learn

Protocols

- Protocols are so totally cool
- `@required` and `@optional` are cool
- Subclassing sucks
- Thing A and Thing B don't know about each other — but they both know about Protocol X

Protocols == plugins

- A good architecture is close to a plugin architecture
- Plugins almost come for free

Drivers

- Set of objects that conform to a protocol
- When something needs to happen, each object is asked if they want to handle it
- First one to say yes, gets to handle it
- Example might be tapping a row in a table
- UTIs can be useful

Delegates Rule

- Protocols and delegates are often bffs
- Even if you use blocks, it's the same thing conceptually

Simple Example (and it should always be simple)

- @protocol RSFeedParserDelegate
 - @required
 - (void)feedParserDidComplete:(id)feedParser;
 - @optional
 - (BOOL)feedParser:(id)feedParser
didParseNewsItem:(RSParsedNewsItem *)
newsItem; //return YES to consume newsItem
 - @end

lynxwire in progress

- <http://bitbucket.org/brentsimmons/lynxwire>
- It's some protocols
- But it's an example of stuff that works in multiple apps

Responder Chain

- Two apps use View X
- But those apps have to implement some actions differently
- Responder chain makes this easy and reduces dependencies
- You can send messages up the responder chain via code (not just via IB)

View Controllers and Responder Chain

- UIViewControllers appear in responder chain
- But NSViewControllers don't
- So just insert them — it's easy

Error presenter chain

- Exists on Macs
- Easy to create for iOS

General Rules of Health

- MVC always.
- Always always. Shut up.
- Don't fight the frameworks.
- Do things the way Apple says. Read between the lines and also do things the way they imply.