Production Objective-C

How to build maintainable, long-lived codebases

Paul Shapiro
Co-founder, CTO at Lunarpad

Roadmap

- My journey
- Problem code and its effects
- Principles for building maintainable apps
  - Architecture for low entropy
  - Factoring
  - Modularity
  - Semantics
- Examples and other goodies
- Recap
- Q&A

My journey

- Early days
- Going pro
- Entry into mobile

Gun for hire

- Linux shop, converted to Macs
- New mobile division
- Extracurricular activities

Rubenstein Technology Group
Technology consultancy

Condé Nast Traveler:
Cruise Finder

Condé Nast Traveler:
Italy


Vidal Sassoon:
The Movie

Pentagram Marks


Xlab 2011

Morrison &
Foerster

9/11 Memorial Guide


RobotGalaxy
Coloring & Activity Book

ValoremLaw

2wice:
Merce Cunningham

Leveling up

Couple

Relationship App for Two; Y Combinator grads
One of the first two iOS dev hires

- Met in California
- Full-time remote; big learning experiences
- Modular programming!

Couple

Proving grounds

bondsy

Social bartering app
Lead mobile developer; TechStars grad

- Entirely modular Objective-C codebase
- Two-week iteration cycle

Bondsy

Production experiences

- Sources of inefficiency
- Developer rework
- Outsourcing

Reality of mobile development

Is writing good code disincentivized?

- Non-reusable code gets deleted
- Higher upfront investment

Reality of mobile development

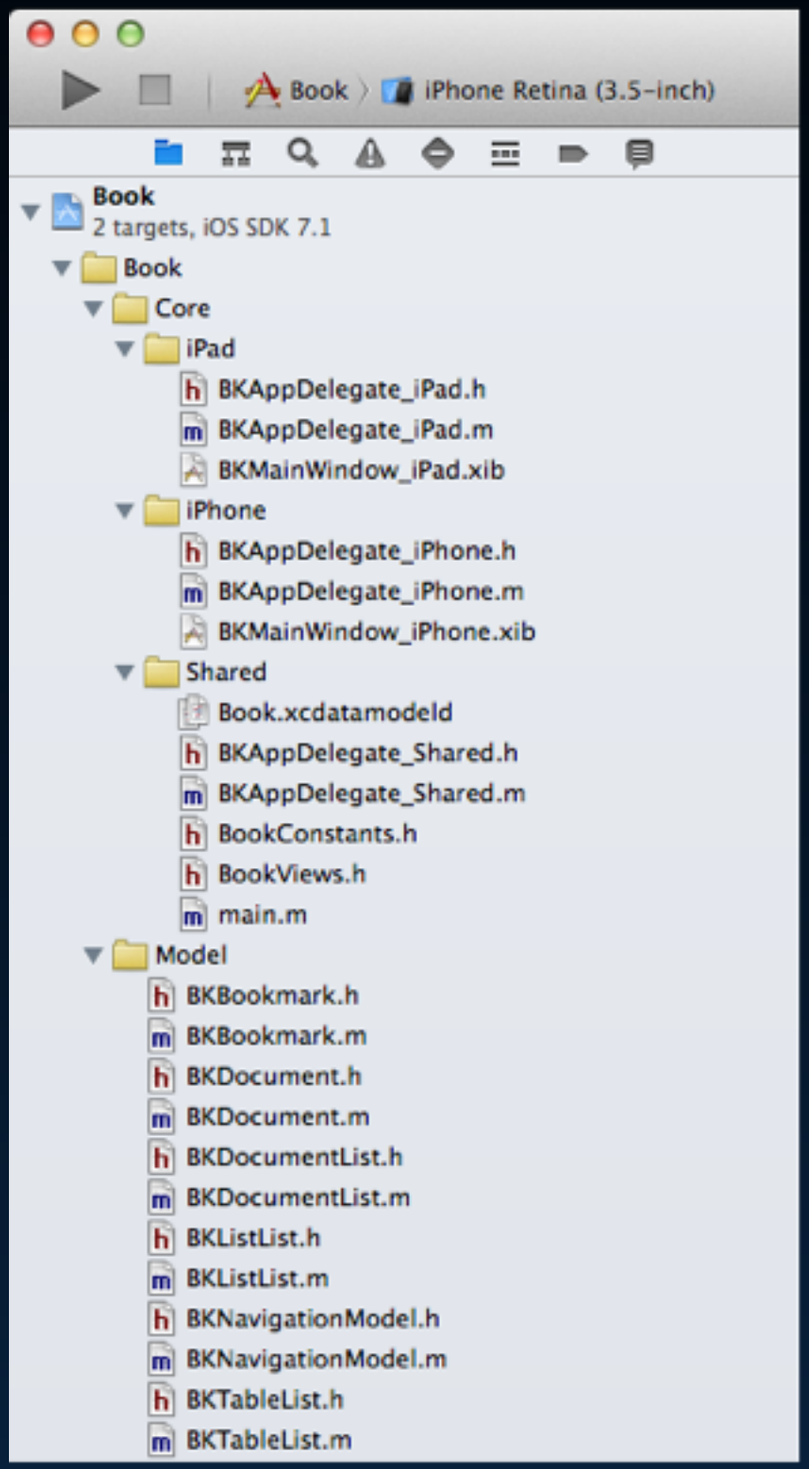
Effects of poor code:

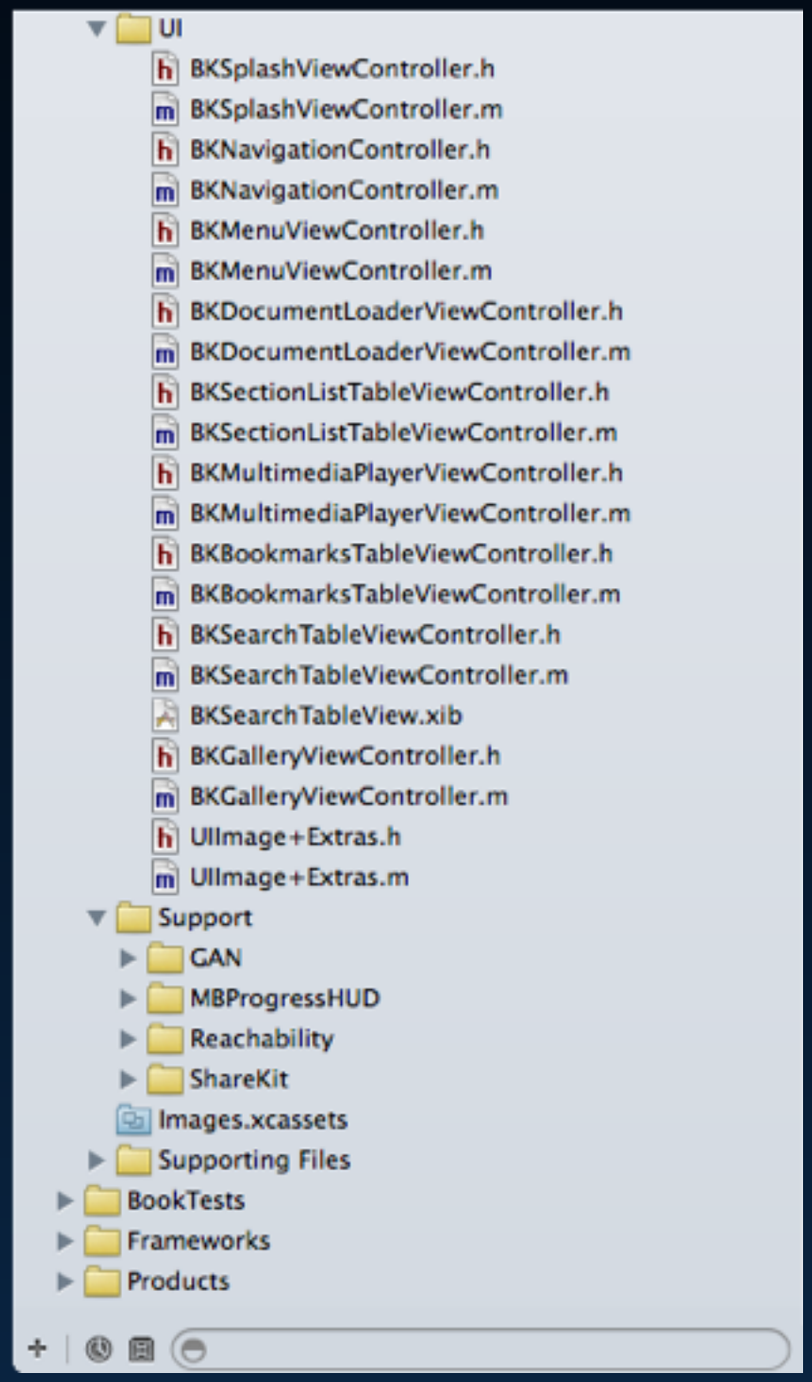
- Hard to test
- Hard for anyone to understand
- Hard to optimize
- Hard to reason about error domains
- Hard to expand
- Usually not reusable nor portable; gets deleted

How are applications built?

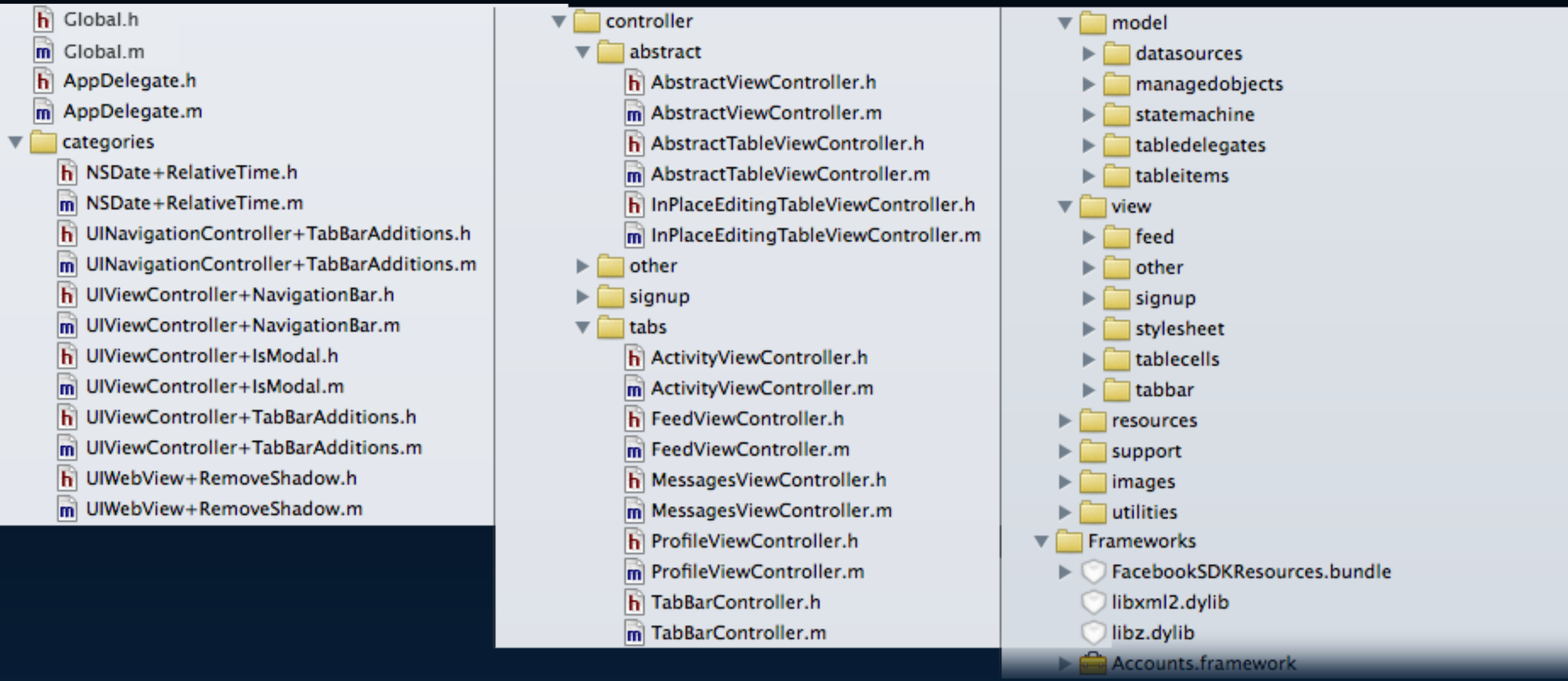
- IDE(s): Xcode, AppCode, …
- SDK: Foundation, UIKit, etc.
- Vendor libraries:
  included directly, as submodules, or via CocoaPods
- MVC application design pattern

Sample content app





Inheriting a problem codebase



Inheriting a problem codebase:
What is an ‘abstract’ table view controller?

```
- (void)tryStartNewPost
{
    TabBarController *tbc = (TabBarController *)[[TTNavigator navigator] rootViewController];
    UINavigationController *nc = (UINavigationController *)[[tbc viewControllers] objectAtIndex:TAB_INDEX_FEED];
    FeedViewController *vc = (FeedViewController *)[[nc viewControllers] objectAtIndex:0];

    if (vc == self) {
        [vc startNewPost];
    }
    else {
        Tab *feedTab = (Tab *)[[[tbc tabBar] tabs] objectAtIndex:TAB_INDEX_FEED];
        [[tbc tabBar] tabSelected:feedTab];

        double delayInSeconds = 0.1f;

        dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, delayInSeconds * NSEC_PER_SEC);
        dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
            [vc startNewPost];
        });
    }
}
```

Inheriting a problem codebase: Factoring the feed table view

```
- (void)startNewPost
{
    if (_isCreatingPost) {
        return;
    }

    if (_isUploadingPost) {

        UIAlertView *alertView =
            [[UIAlertView alloc] initWithTitle:BZLocalizedString(@"Upload In Progress", nil)
                                       message:BZLocalizedString(@"Please wait for the current upload to finish.", nil)
                                      delegate:nil
                             cancelButtonTitle:BZLocalizedString(@"OK", nil)
                             otherButtonTitles:nil];
        [alertView show];
        return;
    }

    FeedDataSource *ds = (FeedDataSource *)self.dataSource;
    if (!ds || ![ds isKindOfClass:[FeedDataSource class]]) {
        return;
    }

    AppDelegate *delegate = [Global delegate];
    delegate.isLinkingTemporarilyDisabled = YES;

    _isCreatingPost = YES;

    [self expand];

    [ds addNewPostInTableView:self.tableView];

    [self.tableView setScrollEnabled:NO];

    [self invalidateView];
```

Inheriting a problem codebase: Data source cruft

```
@interface ListDataSource : TTListDataSource

@end
```

```
@interface UnpagedDataSource : ListDataSource
{
    BOOL _isDataSourceLoading;
    BOOL _isDataSourceLoaded;

    NSMutableArray * _mappedIds;

    NSMutableArray * _delegates;
}

@property (assign, nonatomic) BOOL isDataSourceLoading;
@property (assign, nonatomic) BOOL isDataSourceLoaded;

@property (strong, nonatomic) NSMutableArray * mappedIds;

- (BOOL)shouldLoadMore:(BOOL)more;

@end
```

```
@interface PagedDataSource : ListDataSource
{
    NSUInteger _currentPage;
    NSUInteger _perPage;
    NSUInteger _totalItems;

    NSMutableArray * _mappedIds;

    BOOL _hitLastPage;

    BOOL _isDataSourceLoading;
    BOOL _isDataSourceLoaded;

    NSMutableArray * _delegates;
}

@property (assign, nonatomic) NSUInteger currentPage;
@property (assign, nonatomic) NSUInteger perPage;
@property (assign, nonatomic) NSUInteger totalItems;
@property (assign, nonatomic) BOOL hitLastPage;
@property (assign, nonatomic) BOOL isDataSourceLoading;
@property (assign, nonatomic) BOOL isDataSourceLoaded;

@property (strong, nonatomic) NSMutableArray * mappedIds;

- (BOOL)shouldLoadMore:(BOOL)more;

@end
```

Inheriting a problem codebase: Factoring PagedDataSource

```
- (BOOL)shouldLoadMore:(BOOL)more
{
    if (self.isDataSourceLoading) {
        return NO;
    }

    self.isDataSourceLoading = YES;

    if (more) {
        self.currentPage++;
    }

    if (self.perPage > 0) {
        NSUInteger totalPages = ceilf((float)self.totalItems / self.perPage);

        if ( self.currentPage > totalPages ) {

            self.isDataSourceLoading = NO;

            self.hitLastPage = YES;

            __weak BZPagedDataSource *bself = self;

            int64_t delayInSeconds = 0.1;
            dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, delayInSeconds * NSEC_PER_SEC);
            dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
                [bself.delegates perform:@selector(modelDidFinishLoad:) withObject:self];
            });

            return NO;
        }
    }

    return YES;
}
```

Inheriting a problem codebase: “Utils” classes

```
@interface ImageUtilities : NSObject

+ (UIImage *)resizeImage:(UIImage *)image toSize:(CGSize)newSize;
+ (UIImage *)cropFromCenterImage:(UIImage *)image toSize:(CGRect)size;
+ (UIImage *)resizeAndCropFromCenterImage:(UIImage *)image toSize:(CGSize)size;
+ (UIImage *)cropImage:(UIImage *)image rect:(CGRect)rect;

+ (void)writeJPEG:(UIImage *)image toPath:(NSString *)path;
+ (void)writePNG:(UIImage *)image toPath:(NSString *)path;

+ (UIImage *)imageFromCompositeView:(UIView *)compositeView;

+ (UIImage *)image:(UIImage *)image maskedWithColor:(UIColor *)color;

+ (UIImage *)transparentBorderImage:(UIImage *)image borderSize:(NSUInteger)borderSize;

+ (UIImage *)convertToGreyscale:(UIImage *)i;

+ (UIImage *)fixOrientation:(UIImage *)image;
+ (UIImage *)fixOrientation:(UIImage *)image orientation:(UIImageOrientation)imageOrientation;

@end
```

Analysis

- Poor code quality
- Inconsistent project/object organization
- Dangerous dependencies

Conclusions

- Too many projects are rushed
- Their code often has to be scrapped

Cruft

“I've found in my long career as a slob that **cruft breeds cruft**, and I've seen this happen in software as well as under beds and in the corners of rooms.”

Paul Graham, The Hundred-Year Language

How can we build maintainable
systems from the start?

Cause and effect
Every decision that goes into a system design influences the design complexity, which controls future design decisions.

software entropy
The tendency of a complex system to become more complex through changes

Architecture:
Useful terms

complex
From Latin *com-* (“together”) and *plectere* (“to weave, braid”). See *complect*

simple
From the same root as *semel* (“one”) and *plicō* (“fold”); having a single layer

Architecture:
What causes code complexity?

to complect (obs.)
to interweave; intertwine

to conflate
to combine or blend

Architecture:
Why do we use the MVC pattern in applications?

Model

- data persistence
- lookups/caching
- business logic
- event broadcast/delegation

View

- subview data binding
- user input proxy

Controller

- model↔view integrator
- event router
- imperative executor

Architecture:
Object coupling

- Good and bad coupling
- Problems of bad coupling
- Good coupling requires clear interfaces between properly factored code

Factoring

- What is factoring?
- Factoring makes your life simpler
  - Makes reusable constants, derived values, and implementations
  - Reduces repetition of work
  - Simplifies visualization of relationships

Factoring: Data & its format

Why do we silo in model objects?
e.g. TDToDoItem, TDToDoItemList, BPUser, etc.

- Business logic encapsulation
- Need canonical data source
- Consumer synchronization
- Data I/O formatting
- Serializing for I/O

Siloing: Operations

When implementation repeated, encapsulate within:

- A variable
- A method with argument inputs
- A configurable runtime object, e.g.
  - State machine/runloop/NSOperation
  - A DSL
- A module!

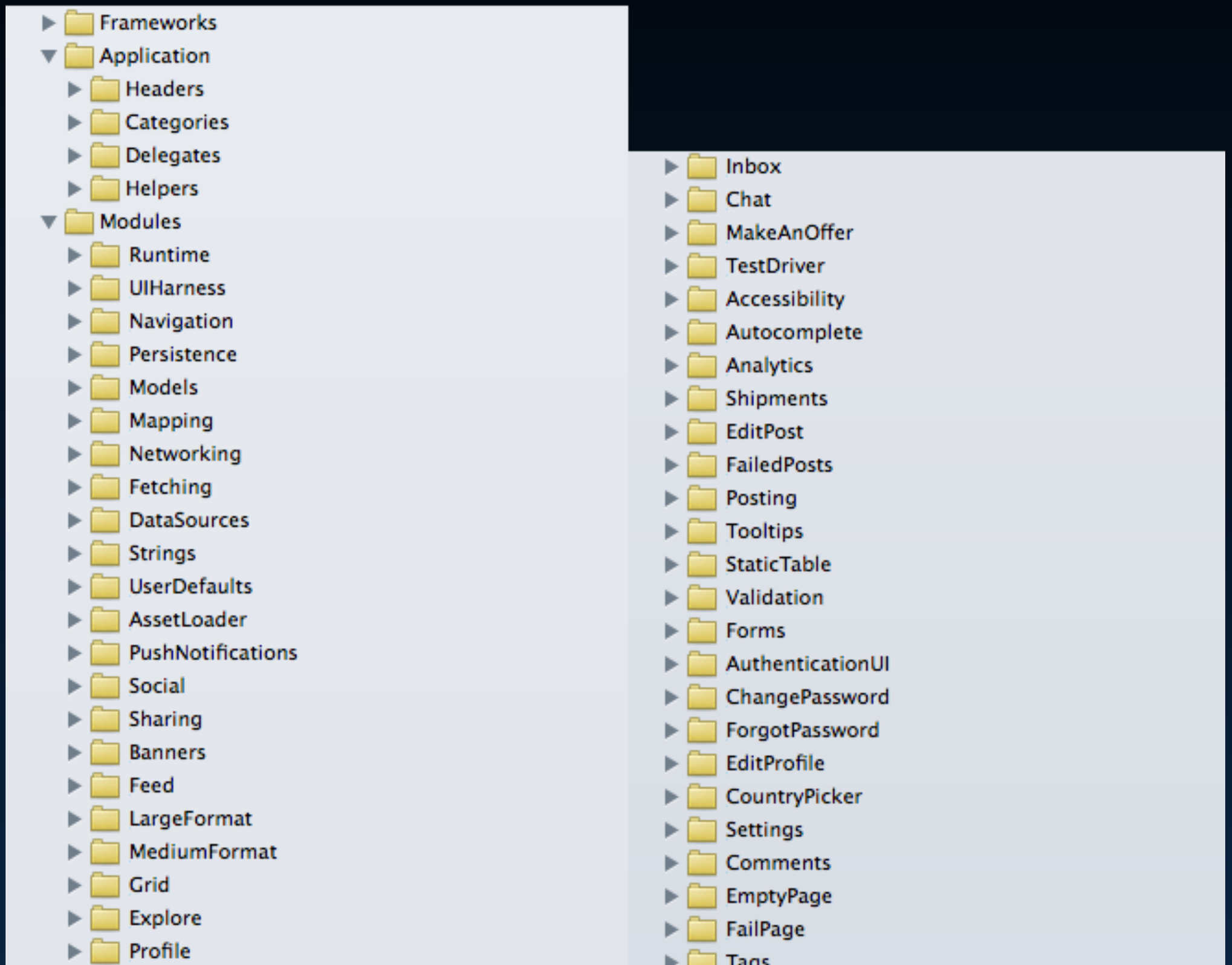
Modules for maintainable code

- **Software module**
  A discrete, meaningful collection of code that is organized into a uniquely named folder

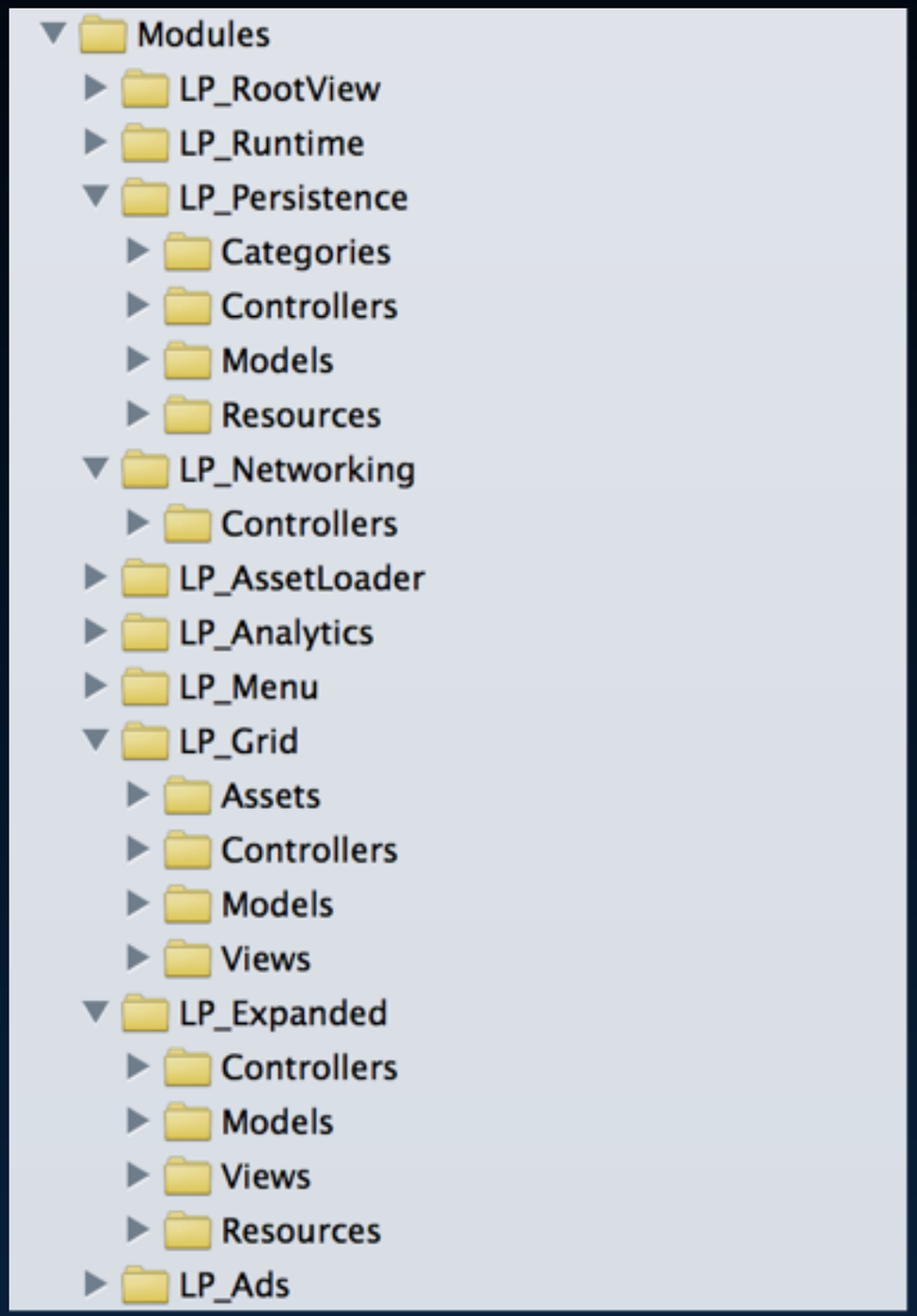
- Scalable codebase expansion

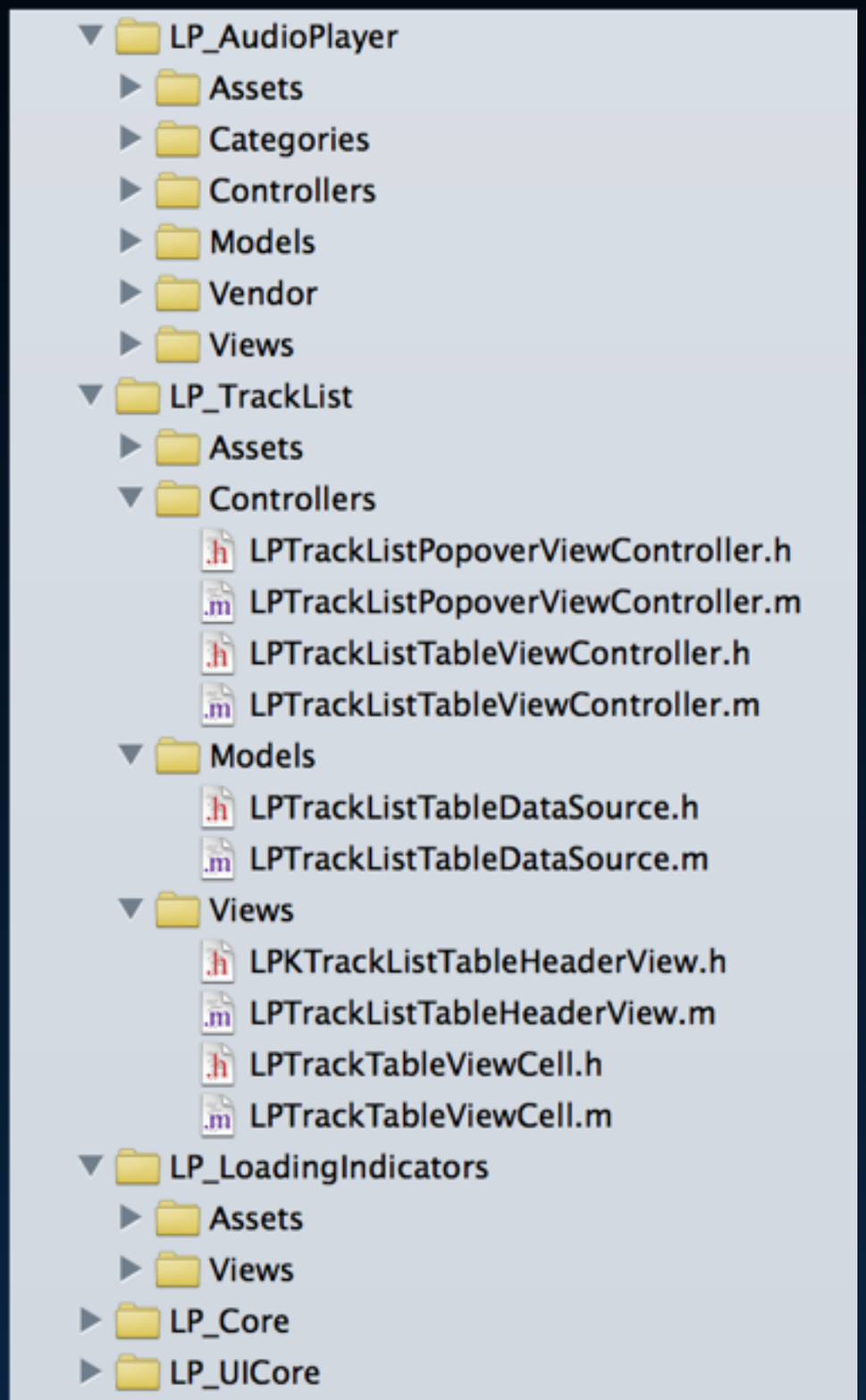
- Services with APIs

A modular codebase



Another modular codebase





Benefits of modular architecture

- Simple conceptualization
- Reliable reasoning
- Incremental progress verification
- Easy code reuse/portability
- Isolated testing
- Merge conflict avoidance

Intra-module connections

Stronger coupling is less dangerous within modules.

- **Delegation**
- **KVO**
- **Blocks**

```
- (id)initWithSetupBlock:(void (^)(id self))setupBlock;
```

Cross-module connections

Low coupling is essential.

- Singletons harder to port; rarely ok.
- Dependency injection (pass on init)
- Dependency imperatives & accessors via @interfaces

Cross-module: public interfaces

- **`@interfaces`:**

  Choose publicly exposed methods and properties carefully

- **`@protocols`**

```
@protocol ANAudioPlayerControlsInteractionsDelegate <NSObject>
- (void)audioPlayerPlayPauseButtonTapped;
- (void)audioPlayerRWButtonTapped;
- (void)audioPlayerFFButtonTapped;
@end
```

Cross-module: public interfaces

- **Constants**

```
extern NSString *const TDToDoList_notification_name_itemWasAdded;
extern NSString *const TDToDoList_notification_userInfo_key_item;
typedef enum { … } TDToDoListItemType;
```

- **Class-level methods for shared constants and public, reusable, anonymous transforms**

```
+ (CGRect)viewFrame;
+ (NSString *)reuseIdentifier;
+ (UIFont *)titleLabelFont;
```

Cross-module connections

- Events: NSNotifications instead of delegation

NSNotifications example (pub/sub)

Emitter/publisher (`OPOperation`):

```
NSDictionary *payload = @
{
      OPOperation_notification_userInfo_key_operation : self
};
  [[NSNotificationCenter defaultCenter]
postNotificationName:OPOperation_notification_didFinish
object:payload];
```

NSNotifications example (pub/sub)

Observer/subscriber/consumer:

```
- (void)_startObservingOperations
{ // called by -init chain
  [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(_operationDidFinish:)
name:OPOperation_notification_didFinish object:nil];
}


- (void)_operationDidFinish:(NSNotification *)note
{
  NSDictionary *userInfo = note.userInfo;
  OPOperation *operation = [userInfo
objectForKey:OPOperation_notification_userInfo_key_operation];
  …
}
```

Cross-module connections

- A word about NSNotifications
- Distributed systems corollaries (Pub/sub, req/rep, …)
- NSOperationQueues, et al.
  - Yet another reason to extract model

Writing better code

- Utility of memorization?
- Machines are deterministic, but sometimes buggy
- Technology ought to make our lives better
- “An answer is in the question.”

Writing better code

- Good code can change existing problem code for the better
- Problems mustn’t be swept under the rug
- How to perfect your programming ability
- Disciplined practice stacks up

The role of semantics

- Programming languages are for humans, not machines.
- **Semantics**
  the meaning of a word, phrase, sentence, or text
- Importance of the meaning of words

Good semantics

- Accuracy, precision, rigor
- Matters, substance, context
- Form, function, purpose, role

Benefits of rigorous semantics

- No namespace collision!
- Clarity of architecture
- Write code, not comments
- When combined with modules: relevant code is immediately locatable

Formulating good semantics

- Good semantics via good understanding
- Modules, objects, methods, variables
- All objects:
  1. **read data**
  2. **internally transform data**
  3. **yield data**

Types of object code

Accessors*

Lookups

Transforms

Factories

```
- (BOOL)isPerformingOperation;
```

‘Passive’ (read)
Always returns something

* Larry tells me Apple uses the term ‘accessors’ to include getters *and* setters.

Object code semantics

Imperatives

State changes

Actions

```
- (void)performOperation;
```

‘Active’ (write)
May return something,
like an error, or call a block

Types of object code

Delegation

Event handling

Routing

```
- (void)operationWasPerformed;
```

‘Passive’

Usually has no return value

Naming object methods

Good method names come from reporting what is happening in the code-path.

Accessors
(new*, is*, …)

```
- (NSUInteger)toDoItemsCount;
  // a lookup accessor method on the model

- (NSMutableArray *)_newToDoItemsMutableArray;
  // a private factory accessor method in the model

- (TDToDoItem *)newToDoItemFromDictionary:(NSDictionary *)dictionary;
  // a public transform accessor method
```

Naming object methods

Imperatives

(active: add*/remove*, perform*, configureWith*, layout*)

```
- (void)addToDoItems:(NSArray *)toDoItemsArray;
  // an imperative method on the model
```

Delegation

(passive: did, will, has, was)

```
- (void)_toDoItemWasAdded:(NSNotification *)note;
  // an NSNotification handler in the controller
```

Naming variables & properties

A sound variable name is derived by describing its content (what it is).

```
NSString *const
TDToDoItemsList_notificationName_toDoItemWasAdded
                =
@“TDToDoItemsList_notificationName_toDoItemWasAdded”;


// Called by a model’s -init chain:
- (void)_setupModel
{
  self.toDoItemsMutableArray
        = [self _newToDoItemsMutableArray];
}
```

Naming variables & properties

```
- (void)_setupRightDisclosureButton
{
    UIButton *button = [UIButton
                buttonWithType:UIButtonTypeCustom];
    …
    [self addSubview:button];
    self.rightDisclosureButton = button;
}
```

Naming module objects

Models: what is it?
e.g. TDToDoList, TDToDoItem

Controllers: what is it operating on, and what does it do?
e.g. TDToDoListTableViewController, TDPersistenceController, TDNetworkingController

Views: what is it displaying? what is it receiving?
e.g. TDToDoItemCell, TDNetworkedImageView, TDPostButton

Human-friendly code formatting

- Separators: two newlines, then 80 forward slashes, then one newline

- `#pragma mark`s after separators with descriptive categories

- Rapid coding via standardization; keyboard shortcuts

Formatting implementation (.m) files

```
#import "…"


///////////////////////////////////////////////////////////////////////////
#pragma mark - Macros



///////////////////////////////////////////////////////////////////////////
#pragma mark - Constants


///////////////////////////////////////////////////////////////////////////
#pragma mark - Interface

@interface LPImageResizingScrollView ()

…

@end
```

Formatting implementation (.m) files

```
////////////////////////////////////////////////////////////////////////
#pragma mark - Implementation

@implementation LPImageResizingScrollView


////////////////////////////////////////////////////////////////////////
#pragma mark - Lifecycle

- (id)init
{
    …
}


////////////////////////////////////////////////////////////////////////
#pragma mark - Accessors - (Factories|Lookups|Transforms)


////////////////////////////////////////////////////////////////////////
#pragma mark - Imperatives


////////////////////////////////////////////////////////////////////////
#pragma mark - Delegation


////////////////////////////////////////////////////////////////////////
#pragma mark - Class

@end
```

Formatting implementation (.m) files

1. Dependencies (`#imports`)
2. Pre-processor macros (`#defines`)
3. C variables (`consts`, `typedefs`, …)
4. C functions (if any)
5. Private interface(s)
6. Implementations
    a. Lifecycle
        `-init*` → `-_setup`
        `-dealloc` → `-_teardown`
    b. Accessors
    c. Imperatives
    d. Delegation
    e. Class-level

Formatting header (.h) files

- Invert pragma mark perspective (usage type instead of code type)
  e.g.

  `#pragma mark - Fetching - Imperatives`

  (instead of `#pragma mark - Imperatives - Fetching`)

xcode-snippets

https://github.com/Lunarpad/xcode-snippets

xcode-snippets

Human-friendly code snippet insertion keyboard shortcuts for Xcode.

Installation:

To install, simply drop these .codesnippet files into `~/Library/Developer/Xcode/UserData/CodeSnippets/`

Manifest:

Shortcut	Function
ddd	Output an 80-character code section separator
pra	Output a separator, and a pragma mark ready to be labeled
acc	Output a separator, and a pragma mark with the label, "Accessors"
imp	Output a separator, and a pragma mark with the label, "Imperatives"
del	Output a separator, and a pragma mark with the label, "Delegation"
int	Output a scaffold for the header of an implementation file
obj	Output a scaffold for starting a new object @implementation internals

Example modules

https://github.com/Lunarpad/Lunarpad-ObjC-Modals

Modals

Description

A view controller module for displaying floating modal views within your iOS app.

A view controller module for displaying floating modal views within your iOS app.

Installation

Simply plop this repo into your project, and you should be good to go.

Sample usage

```
UIViewController *customContentViewController = [[UIViewController alloc] init];
customContentViewController.view.frame = CGRectMake(10, 20, 300, 200); // set intended 'visible' frame
customContentViewController.view.backgroundColor = [UIColor white];
customContentViewController.view.layer.masksToBounds = YES;
customContentViewController.view.layer.cornerRadius = 6;

LPModalViewController *modalViewController = [[BZModalViewController alloc] init];
[modalViewController useContentViewController:myModalContentViewController];

modalViewController.revealTransition = LPModalTransitionSlideUp;
modalViewController.dismissTransition = LPModalTransitionSlideDown;

[modalViewController revealWithCompletion:^(BOOL finished) {}];
```

Example modules

https://github.com/Lunarpad/Lunarpad-ObjC-ImageResizer

ImageResizer

An image-cropping UIScrollView for iOS

Description

This is a UIScrollView subclass that can load an image within its bounds, allow the user to resize the image, and then quickly return to you the cropped image in full original resolution.

Dependencies

- NYXImagesKit

Installation

Simply install dependencies, plop this repo into your project, and you should be good to go.

Example modules

https://github.com/Lunarpad/Lunarpad-ObjC-SegmentedControl

SegmentedControl

Description

A simple segmented control view for iOS 7.

Installation

Simply plop this repo into your project, and you should be good to go.

Usage

```
@interface LPSegmentedControl : UIView

- (id)initWithFrame:(CGRect)frame andButtonTitles:(NSArray *)buttonTitles andTintColor:(UIColor *)tintColor a

- (void)selectButtonAtIndex:(NSUInteger)buttonIndex;

@end



@protocol LPSegmentedControlDelegate <NSObject>

- (void)segmentedControl:(LPSegmentedControl *)segmentedControl buttonTappedWithIndex:(NSUInteger)buttonIndex
```

More goodies

https://github.com/Lunarpad/Lunarpad-ObjC-UIViewDebug

UIViewDebug

Description

Objective-C extensions for debugging UIViews.

Installation

Simply plop this repo into your project, and you should be good to go.

Usage

Auto-bordering UIViews

```
UIView *view = [[UIView alloc] init];
[view giveBorder]; // give the view a randomly colored 1px border
[view borderSubviews]; // random-color borders all subviews
```

More goodies

https://github.com/Lunarpad/Lunarpad-ObjC-Concurrency

Concurrency

Description

Objective-C utility methods for performing Grand Central Dispatch (GCD) concurrency operations, like context switching.

Installation

Simply plop this repo into your project, and you should be good to go.

Available methods in `LPConcurrency`

```
+ (void)performBlockOnMainQueue:(void(^)(void))block; // async
+ (void)performBlockOnMainQueueSynchronously:(void(^)(void))block; // sync
+ (void)performBlockOnMainQueue:(void(^)(void))block afterDelay:(NSTimeInterval)delay;

+ (void)performBlockInBackground:(void(^)(void))block;

+ (void)performBlockSynchronously:(void(^)(void))block;
+ (void)performBlockSynchronouslyOnMainQueue:(void(^)(void))block;
```

More goodies

https://github.com/Lunarpad/Lunarpad-ObjC-Notifications

Notifications

Description

C/Objective-C convenience methods that wrap NSNotificationCenter

Useful for easy NSNotification posting.

Installation

Simply plop this repo into your project, and you should be good to go.

Usage

```
LPNotificationCenterPost(LPObject_notification_name_sampleEventDidOccur);
```

Sending a payload

```
NSDictionary *userInfo = @
{
    LPObject_notification_userInfo_key_sampleKey : @"value"
};
LPNotificationCenterPostWithInfo(LPObject_notificationName_sampleEventDidOccur, userInfo);
```

What we're doing at Lunarpad

Producer

The first tool that enables designers to make truly custom, native iOS apps without writing any code.

Learn more at **lunarpad.com/products**

Lunarpad is hiring!

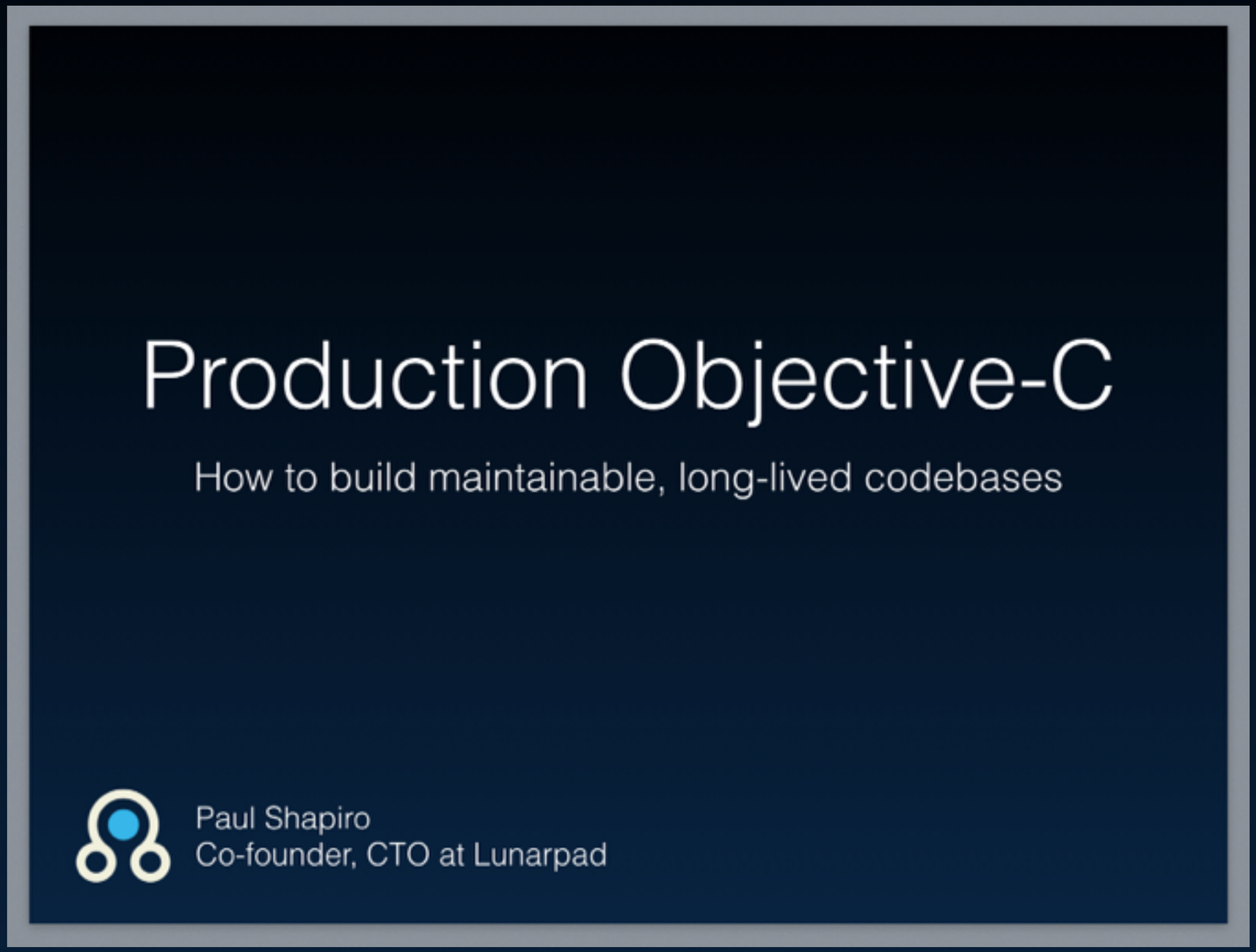
- Browser-side: HTML5/JS
- Back-end: Node.JS
- iOS: Objective-C/C
- OS X
- Linux
- Android

- We work on good problems: compilers, languages, distributed systems, security, virtualization, graphics, *and more!*
- Bootstrapped & private: we answer only to our customers
- Team of very good people
- Full benefits; unlimited vacation; setup of choice; summertime BBQs; home-cooked meals

Recap

- Problem code and its effects
- Principles for building maintainable apps
  - Architecture for low entropy
    - Factoring
    - Modularity
    - Semantics
- Examples and other goodies

This talk



www.lunarpad.com/objc-talk.pdf

Thank you!

Production Objective-C

Paul Shapiro
@tweetingpauls
paul@lunarpad.com