Keepin' It Functional

Automated UI Testing using KIF!

Conrad Stoll
Architect
mutualmobile

Mutual Mobile

The mobile solutions company.

Demo

Agenda

What are we doing here?

- Why we should automate
- KIF
- Writing Tests with KIF
- KIF Testing Philosophy
- Efficiency with KIF
- Another Demo!
- Questions?

Why We Should Automate

Why we should automate

“Automate ALL THE THINGS!” - Jonathan Penn

Automated testing helps us provide value to our users.

- Notification of failure.
- Identify problems in our UI.
- Build better apps.
- Make our apps accessible.
- Improve performance.

KIF

Intro to KIF

Ok, so what the heck is this and how does it work?

- Origins of KIF
- Anatomy of a KIF Test
- Future of KIF

Origins of KIF

In the beginning, there was Square.

Square created KIF to help ensure their apps were providing value to their users, e.g. enabling them to earn a living and buy food.

- KIF was released by Square in 2011.
- Square built KIF in one month.
- Square wants KIF to be simple.
- Maintainable by other iOS engineers.

Why we chose KIF

To use square brackets, or not to use square brackets?

At Mutual Mobile, we employ 60 Objective-C developers. Using the same language to write our apps and our tests is more efficient.

- Part of our development process.
- Integrated in our CI environment.
- Regression testing.
- Acceptance testing.
- Getting up and running with KIF is easy.
- KIF is very simple, and getting simpler.

Writing KIF Tests

Anatomy of a KIF Test

Pieces of the puzzle

A KIF test is a scenario that a user would perform which is composed of a series of repeatable steps.

- KIFTestStep
- KIFTestScenario
- KIFTestController

KIFTestStep

The building block of Automation

KIFTestStep represents a single action that a user could perform, such as tapping a button, or scrolling a table.

- Use accessibility labels for identification.
- KIF provides many useful pre-defined steps.
- Represents an action the user is taking.
- Should have a good description.

KIFTestStep

The building block of Automation

```
+ (id)stepToWaitForViewWithAccessibilityLabel:(NSString *)label;
+ (id)stepToWaitForAbsenceOfViewWithAccessibilityLabel:(NSString *)label;
+ (id)stepToWaitForTappableViewWithAccessibilityLabel:(NSString *)label;
+ (id)stepToWaitForNotificationName:(NSString*)name object:(id)object;

+ (id)stepToTapViewWithAccessibilityLabel:(NSString *)label;
+ (id)stepToTapScreenAtPoint:(CGPoint)screenPoint;
+ (id)stepToTapRowInTableViewWithAccessibilityLabel:(NSString*)tableViewLabel atIndexPath:(NSIndexPath *)indexPath;

+ (id)stepToLongPressViewWithAccessibilityLabel:(NSString *)label duration:(NSTimeInterval)duration;
+ (id)stepToEnterText:(NSString *)text intoViewWithAccessibilityLabel:(NSString *)label;
+ (id)stepToSwipeViewWithAccessibilityLabel:(NSString *)label inDirection:(KIFSwipeDirection)direction;
```

KIFTestScenario

The warhorse of Automation

KIFTestScenario represents a set of user actions (steps) that create a use case scenario.

- Method name should describe workflow.
- Should run from a known state.
- Should be repeatable.
- The order of steps is static.
- Should not depend on each other...
- ... most of the time.

KIFTestScenario

The warhorse of Automation

```
+ (id)scenarioWithDescription:(NSString *)description;

+ (void)setDefaultStepsToSetUp:(NSArray *)steps;
+ (void)setDefaultStepsToTearDown:(NSArray *)steps;

- (void)addStep:(KIFTestStep *)step;
- (void)addStepsFromArray:(NSArray *)steps;
```

KIFTestController

The General of Automation

KIFTestController is responsible for executing your test scenarios. It owns both the output of results and the order in which scenarios are executed.

- Your test controller can be very basic.
- You should create a custom controller.
- Helps you organize your tests.
- Helps you write cleaner test code.

Writing Tests with KIF

Ok fine, here's some sample code.

```
+ (NSArray *)stepsToCreateNewMeme {
    NSMutableArray *steps = [NSMutableArray array];
    [steps addObjectsFromArray:[self navigationStepsToTextEntryScreen]];
    [steps addObjectsFromArray:[self stepsToEnterTextAndMoveToTextPlacementScreen]];
    [steps addObject:[KIFTestStep stepToTapViewWithAccessibilityLabel:@"Save"]];
    [steps addObject:[KIFTestStep stepToWaitForViewWithAccessibilityLabel:@"meme"]];
    return steps;
}

+ (KIFTestScenario *)scenarioToTestMemeCreation {
    KIFTestScenario *scenario = [KIFTestScenario scenarioWithDescription:@"Test Meme
Creation Workflow"];
    [scenario addStepsFromArray:[self stepsToCreateNewMeme]];
    return scenario;
}
```

Writing Tests with KIF

Ok fine, here's some sample code.

```
- (void)addStepsToStartRunStopwatch {
    [self addStep:[KIFTestStep stepToTapViewWithAccessibilityLabel:@"Add"]];
    [self addStep:[KIFTestStep stepToWaitForViewWithAccessibilityLabel:@"Stopwatch"]];
    [self addStep:[KIFTestStep stepToTapViewWithAccessibilityLabel:@"Stopwatch"]];
    [self addStep:
        [KIFTestStep stepToWaitForViewWithAccessibilityLabel:@"stopwatch start"]];
    [self addStep:
        [KIFTestStep stepToTapViewWithAccessibilityLabel:@"stopwatch start"]];
}

- (void)addStepsToSaveRun {
    [self addStep:[KIFTestStep stepToTapViewWithAccessibilityLabel:@"Save"]];
    [self addStep:
        [KIFTestStep stepToWaitForTimeInterval:1 description:@"Wait for Alert"]];
    [self addStep:[KIFTestStep stepToTapViewWithAccessibilityLabel:@"Save"]];
    [self addStep:[KIFTestStep stepToWaitForViewWithAccessibilityLabel:@"Runs"]];
}
```

KIF Testing Philosophy

What We Should Test with KIF

Wait, don't we test EVERYTHING with KIF?

Think about what you want your tests to tell you.

- Test your app broadly.
- Tests should be simple.
- Performance.
- General use cases.
- Repeatable use cases.
- Regression.
- Acceptance.

What We Should NOT Test with KIF

Wait, don't we test EVERYTHING with KIF?

Make sure you're testing the right things with the right tool.

- NOT for testing web services.
- NOT ideal for edge cases.
- Don't forget about unit tests.

Efficiency with KIF

Keeping your Code Clean

Avoiding the Category Spaghetti Monster.

KIF steps and scenarios are created using categories by convention. While that is convenient, that can get hairy as your tests start to multiply.

- Remember SOLID.
- Reuse steps and parts of scenarios.
- Give your scenarios and steps good names.

Organizing Your Tests

Or at least make them predictably cluttered.

Creating factories for each section of your app that vend scenarios maximizes reuse and readability.

- Scenario factories.
- Step factories that add steps to a scenario.
- Add scenarios by name in test controller.
- Create a factory for each functional area.

Organizing Your Tests

Or at least make them predictably cluttered.

```
@interface WLRunScenarioFactory : NSObject

+ (KIFTestScenario *)scenarioToAddRunNamedMyRun;
+ (KIFTestScenario *)scenarioToAddAndVerifyRunNamedMyRun;
+ (KIFTestScenario *)scenarioToAddRunNamedMyRunAndRenameToTownLake;
+ (KIFTestScenario *)scenarioToAddRunNamedToonLakeAndDeleteIt;
+ (KIFTestScenario *)scenarioToAddUnNamedRunAndGoForARunAndViewMapAndGetStuck:
    (BOOL)getStuck;
+ (KIFTestScenario *)scenarioToAddRunNamedGoAndGoForARunThenReNameRunAndViewMap;
+ (KIFTestScenario *)scenarioToAddRunNamedMyRunAndGoForARunAndVerifyNotes;
+ (KIFTestScenario *)
    scenarioToAddRunNamedMyRunAndGoForARunAndFailToEnterNotesWhileRunning;

@end
```

Organizing Your Tests

Or at least make them predictably cluttered.

```
@implementation WLRunScenarioFactory
+ (KIFTestScenario *)scenarioToAddRunNamedMyRun {
    KIFTestScenario *scenario =
        [self baseScenarioWithDescription:@"Add run named My Run."];
    [scenario addStepsToAddRunWithName:@"My Run" description:@"My Description"];
    return scenario;
}

+ (KIFTestScenario *)scenarioToAddAndVerifyRunNamedMyRun {
    KIFTestScenario *scenario =
        [self baseScenarioWithDescription:@"Add run named My Run."];
    [scenario addStepsToAddRunWithName:@"My Run" description:@"My Description"];
    [scenario addStepsToViewRunWithName:@"My Run"
                            description:@"My Description"
                            atIndexPath:[NSIndexPath indexPathForRow:0 inSection:0]];
    return scenario;
}
```

Organizing Your Tests

Or at least make them predictably cluttered.

```
@interface KIFTestScenario (WLRunStepFactory)

- (void)addStepsToAddRunWithName:(NSString *)runName
                     description:(NSString *)runDescription;
- (void)addStepsToViewRunWithName:(NSString *)runName
                      description:(NSString *)runDescription
                      atIndexPath:(NSIndexPath *)indexPath;
- (void)addStepsToEditRunNameFrom:(NSString *)fromName toName:(NSString *)toName;
- (void)addStepsToDeleteRunWithName:(NSString *)runName
                        description:(NSString *)runDescription
                        atIndexPath:(NSIndexPath *)indexPath;
- (void)addStepsToEnterNoteOnRunStopwatchWithText:(NSString *)noteText; // Requires stopwatch to be stopped
- (void)addStepsToStartRunStopwatch;
- (void)addStepsToStopStopwatch;
- (void)addStepsToSaveRun;
- (void)addStepsToStopAndSaveRun;
- (void)addStepsToStopAndCancelRun;
- (void)addStepsToViewMapOfRecentRunAtDateString:(NSString *)dateString;
- (void)addStepsToViewNotesFromRunTimeDetail;

- (void)addStepToTapDoneButton;

@end
```

Organizing Your Tests

Or at least make them predictably cluttered.

```
@implementation KIFTestScenario (WLRunStepFactory)
- (void)addStepsToViewRunWithName:(NSString *)runName
                      description:(NSString *)runDescription
                      atIndexPath:(NSIndexPath *)indexPath {
    NSString *label = runName;

    if (runDescription) {
        label = [NSString stringWithFormat:@"%@, %@", runName, runDescription];
    }

    [self addStep:[KIFTestStep stepToTapRowInTableViewWithAccessibilityLabel:label
                                                                 atIndexPath:indexPath]];
    [self addStep:[KIFTestStep stepToWaitForViewWithAccessibilityLabel:
         [NSString stringWithFormat:@"Title, %@", runName]]];
    [self addStep:[KIFTestStep stepToWaitForViewWithAccessibilityLabel:runName]];
}
```

Code Snippets

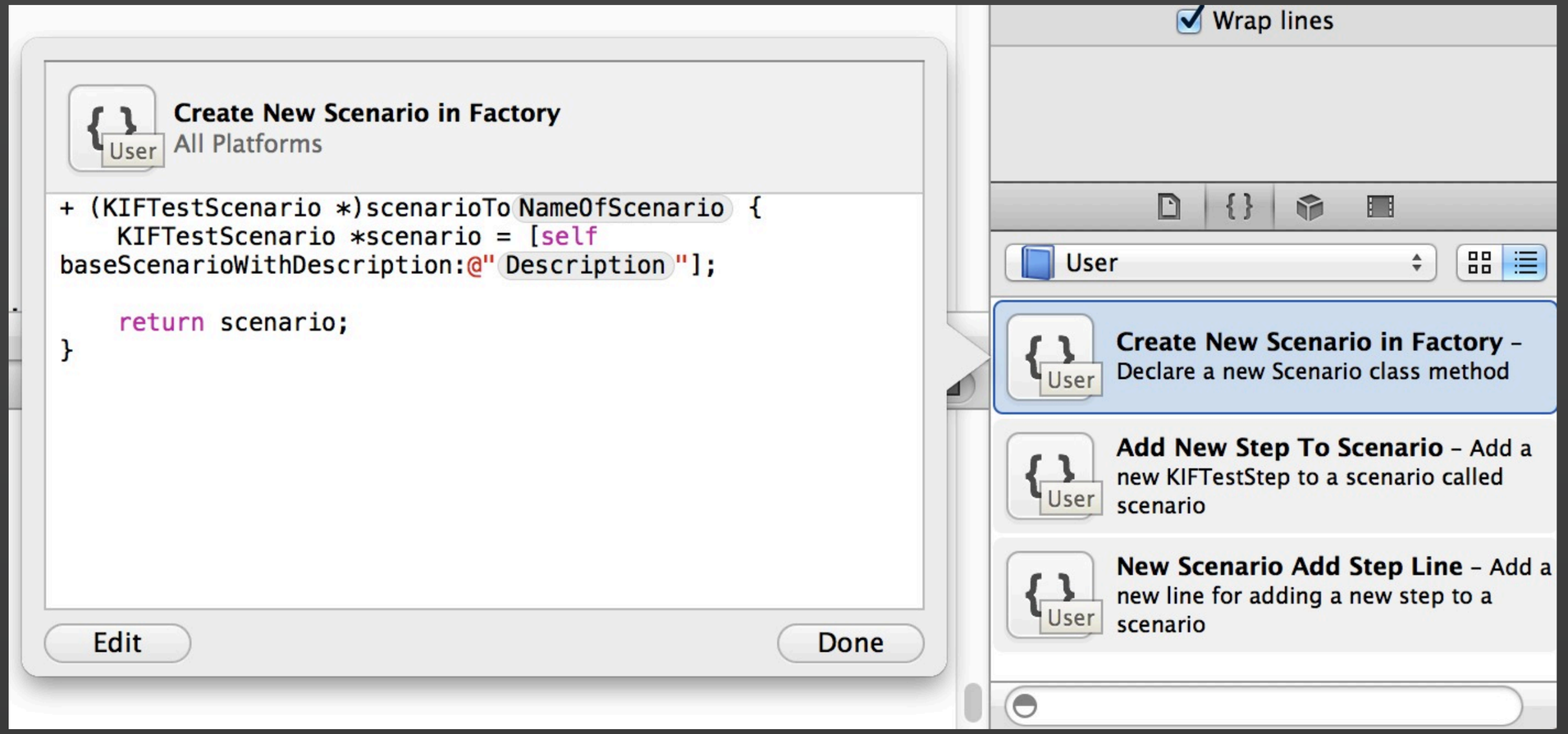
or: How I learned to stop worrying and love the long method names.

Code Snippets in Xcode are a really efficient way to work with very template driven workflows like KIF testing. The very standard KIF naming conventions lends it very naturally to using Code Snippets.

- Snippet for creating a scenario.
- Snippet for adding steps to scenario.

Code Snippets

or: How I learned to stop worrying and love the long method names.



Code Snippets

or: How I learned to stop worrying and love the long method names.

```
+ (KIFTestScenario *)scenarioToNameOfScenario {
    KIFTestScenario *scenario = [self baseScenarioWithDescription:@"Description"];

    return scenario;
}


+ (KIFTestScenario *)scenarioToCreateARun {
    KIFTestScenario *scenario = [self baseScenarioWithDescription:@"Create a Run"];
     [scenario addStep:[KIFTestStep stepMethodSignature]];
    return scenario;
}
```

Future of KIF

Future of KIF

Ok, so what's next?

The next version of KIF is currently in development. The goal is to make KIF even simpler.

- KIF-next was started in community.
- Square is working on finishing it.
- KIFTestController is gone.
- Replaced by SenTestKit.
- Why reinvent the wheel?

Extending KIF

What else can we make this do?

KIF is very easy to extend. We can easily add new steps, convenience methods, and better test result output.

- Add steps for custom views.
- Add convenient custom steps.
- SHORTER METHOD NAMES.
- Additions don’t require base changes.
- Improve the test output with JUnit.
- https://github.com/mutualmobile/KIF

Extending KIF

Why we should extend it

If you find yourself building a lot of KIF tests, you really should extend it.

- Helps you begin to own your own tools.
- Keep your code clean.
- Makes you more efficient.
- Reuse your extensions.

Demo!

Live coding…automatic style!

Learn More?

Where can I go for help?

There’s a lot of resources out there for learning about KIF and automated UI testing.

- https://github.com/square/KIF
- https://groups.google.com/group/kif-framework
- Test iOS Apps with UI Automation - Jonathan Penn

Questions?

Is he serious about us testing our apps?

Thanks!!

Go forth and create user value!

Conrad Stoll

@conradstoll - cnstoll@me.com

mutualmobile