

Unit Testing in Xcode 5

Josh Hinman
Lead iOS Developer, Defy Media
@joshhinman
<http://joshhinman.com>

Types of tests

- Build+Run
- Unit testing
- Integration testing
- Performance
- Usability

Types of tests

- Build+Run
- **Unit testing**
- Integration testing
- Performance
- Usability

A unit test is a piece of code that tests another piece of code.

Monkey.h

```
@interface Monkey : NSObject

@property (nonatomic, getter=isHungry) BOOL hungry;

- (void)eat:(Banana *)banana;

@end
```

Start with a requirement.

“When the monkey eats a banana,
his hunger should be satisfied.”

Testing the Monkey class

Set pre-conditions

```
id b = [OCMockObject mockForClass:[Banana class]];
Monkey *monkey = [[Monkey alloc] init];
monkey.hungry = YES;
```

Exercise the method

```
[monkey eat:b];
```

Check post conditions

```
XCTAssertFalse(monkey.hungry);
```

Unit tests are:

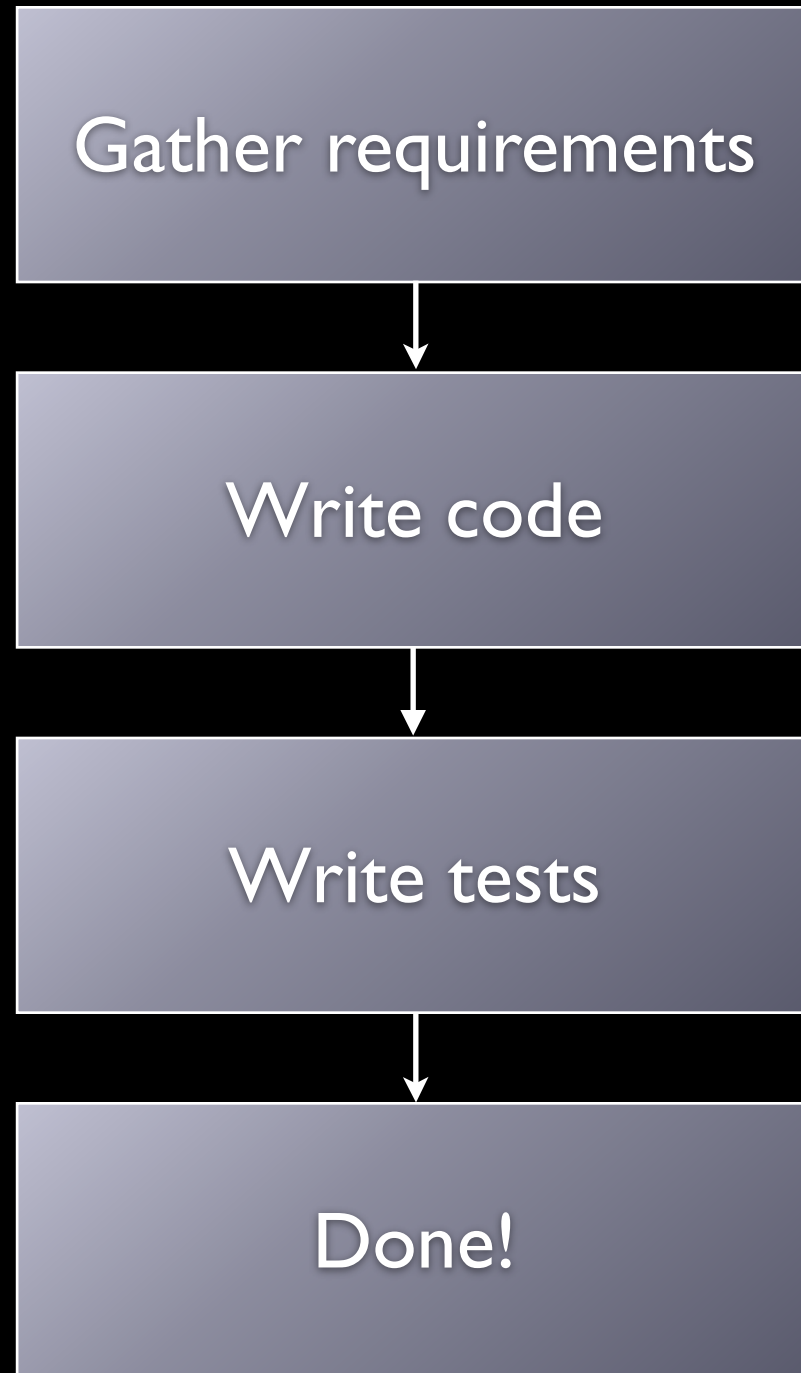
- Isolated
- Automated
 - via continuous integration when possible (Jenkins, Travis CI, Xcode Server)
- Enforce requirements and contracts

Why write tests?

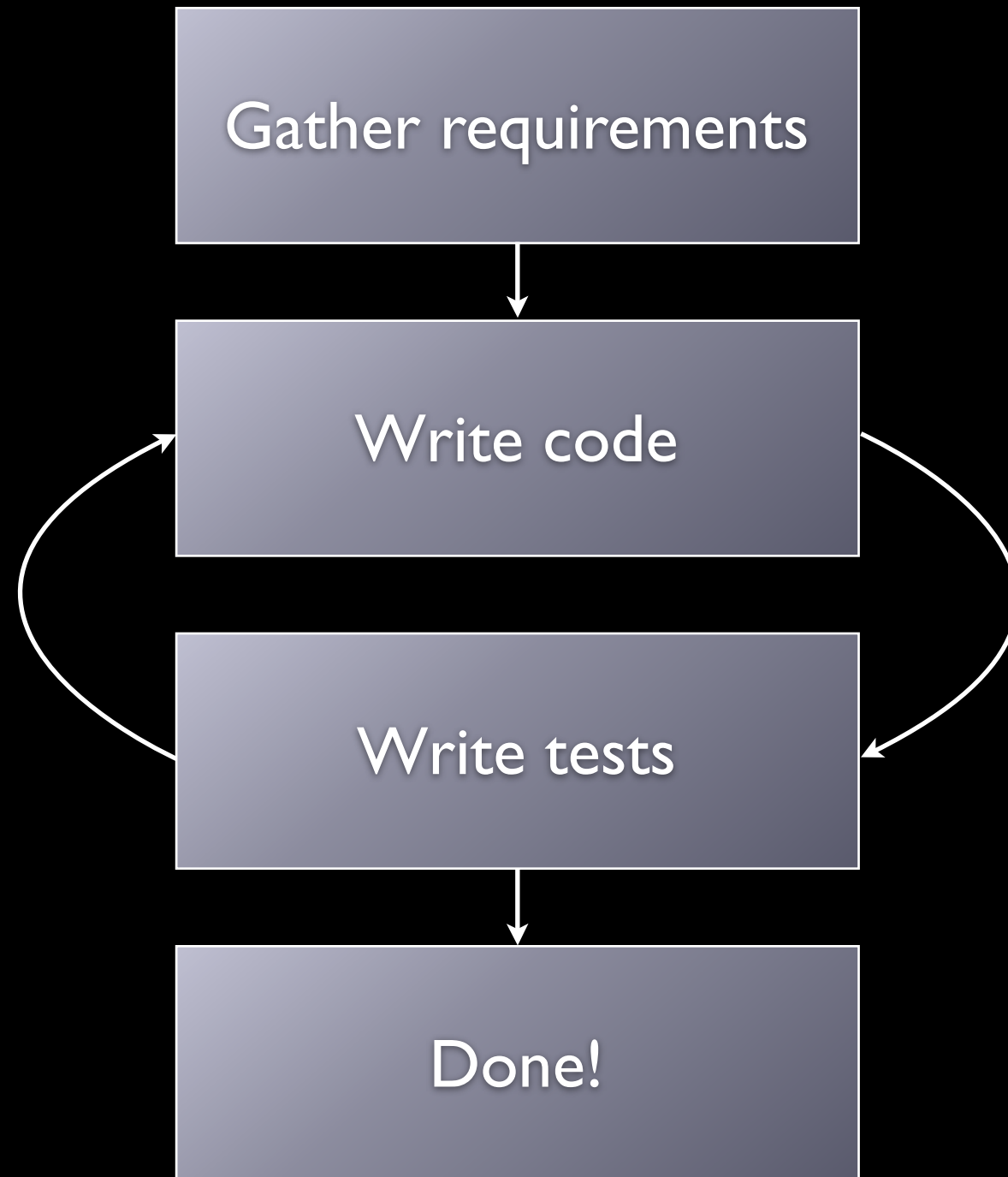
- Isolate problems
 - Including *design* problems!
- Fearless refactoring
- Prevent embarrassing regressions
- Write better code!

Workflow

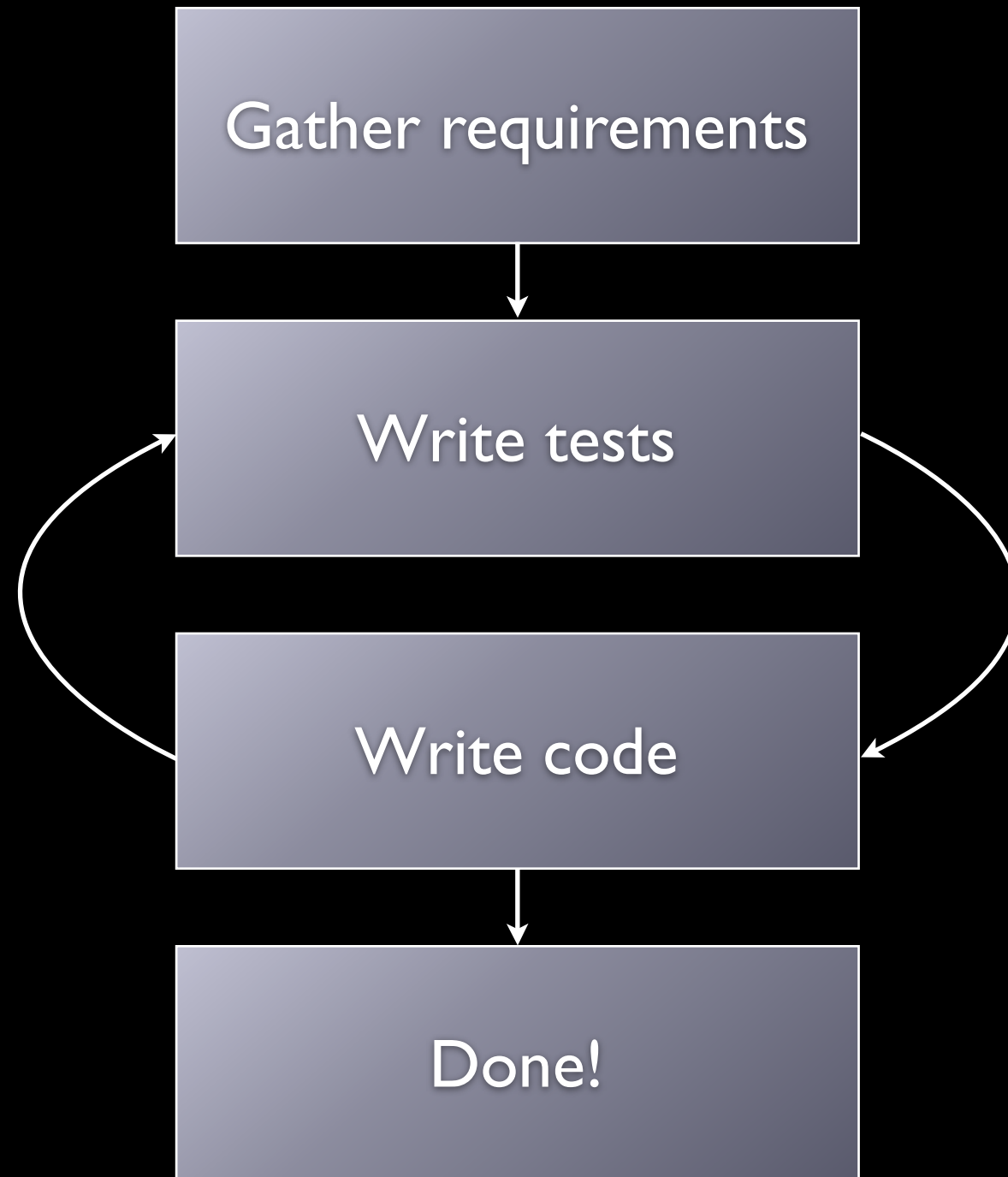
Workflow



Workflow



Test-driven Workflow



Start with a requirement

“When the monkey eats a banana,
his hunger should be satisfied.”

Start with a requirement

```
- (void)testMonkeyEatsBanana
{
    // preconditions
    id b = [OCMockObject mockForClass:[Banana class]];
    Monkey *monkey = [[Monkey alloc] init];
    m.hungry = YES;

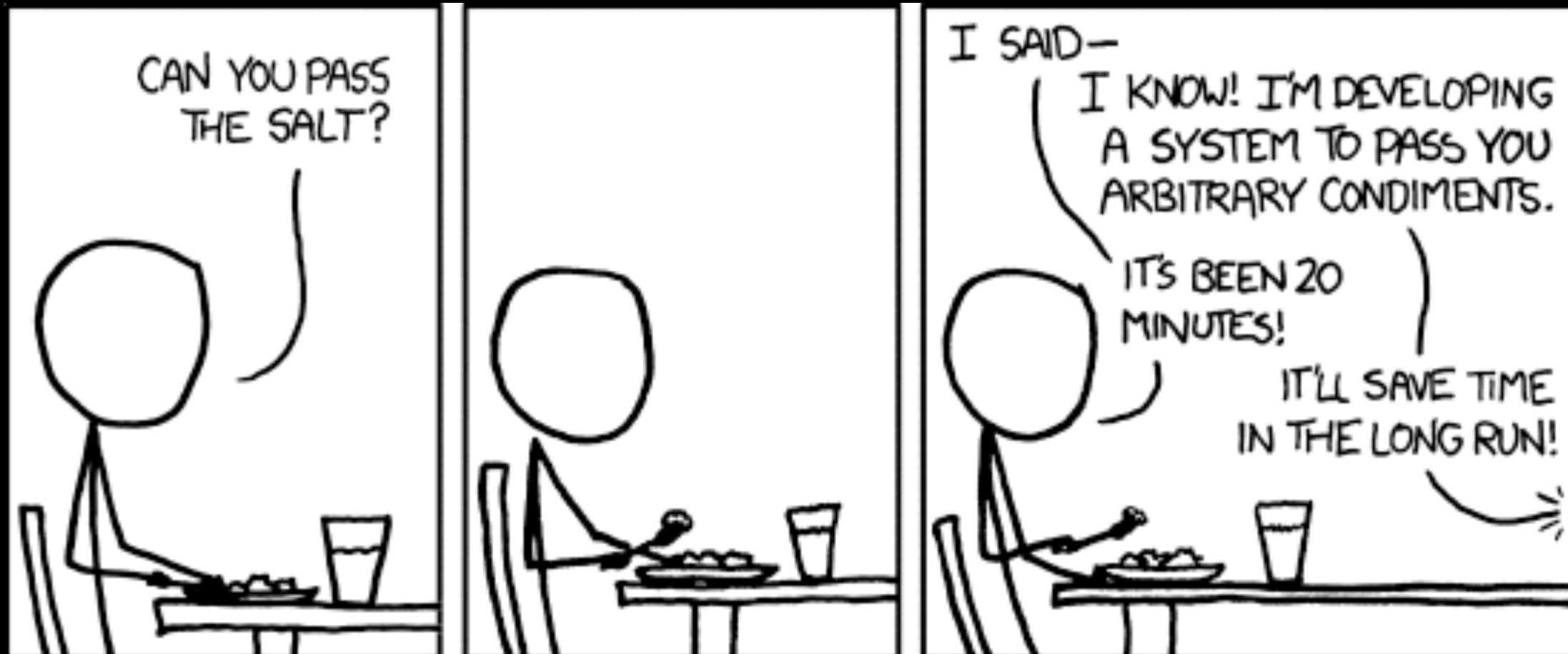
    // test
    [monkey eat:b];

    // postconditions
    XCTAssertFalse(monkey.hungry);
}
```

Benefits of test-first

- YAGNI (you ain't gonna need it)

Obligatory XKCD



Benefits of test-first

- YAGNI (you ain't gonna need it)
 - Everything line of code you write has value

Benefits of test-first

- YAGNI (you ain't gonna need it)
 - Everything line of code you write has value
- Confidence to refactor
 - Red, Green, Refactor
- Early encouragement
- It really is the only way to get good coverage.

Where Unit Tests won't help

- Misunderstood requirements
- UI code is particularly hard to test
- Errors in integration

Demonstration.

More Information



Test-Driven iOS Development

Graham Lee

<http://www.amazon.com/dp/0321774183/>



OCMock Framework

<http://ocmock.org>



Testing in Xcode 5

<https://developer.apple.com/wwdc/videos/?id=409>

The end.

