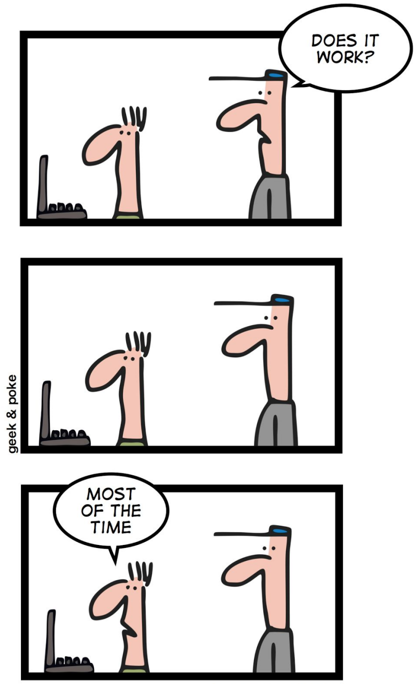
Well-Factored Code [DRAFT]

Or, how to write code that’s easy to Unit Test

SIMPLY EXPLAINED



CONCURRENCY

What is Unit Testing?

Unit Testing is:

- Writing code test cases to validate class behavior
- Can be automated or manual, internal or external
- Not only for finding bugs! Code isolation, contracts, documentation...
- **We already test**, but then throw away the code

Other types of code testing:

- Integration, regression, functional, acceptance, performance, ...
- http://www.testingexcellence.com/types-of-software-testing-complete-list/

What is Unit Testing?

```
// 16 Bit - R=2 G=8 B=14 A=100%

testColor = [UIColor colorFromHexString:@"#28E"];

trueColor = [UIColor colorWithRed:(2.0f / 15.0f)
                            green:(8.0f / 15.0f)
                             blue:(14.0f / 15.0f)
                            alpha:1.0f];

S8Assert([testColor isEqual:trueColor],
    @"[UIColor colorFromHexString] fail: '#28E' differs from TRUE value!");
```

What is Unit Testing?

```
// 16 Bit - R=2 G=8 B=14 A=100%

testColor = [UIColor colorFromHexString:@"#28E"];

trueColor = [UIColor colorWithRed:(2.0f / 15.0f)
                            green:(8.0f / 15.0f)
                             blue:(14.0f / 15.0f)
                            alpha:1.0f];

S8Assert([testColor isEqual:trueColor],
    @"[UIColor colorFromHexString] fail: '#28E' differs from TRUE value!");
```

Well-Factored Code

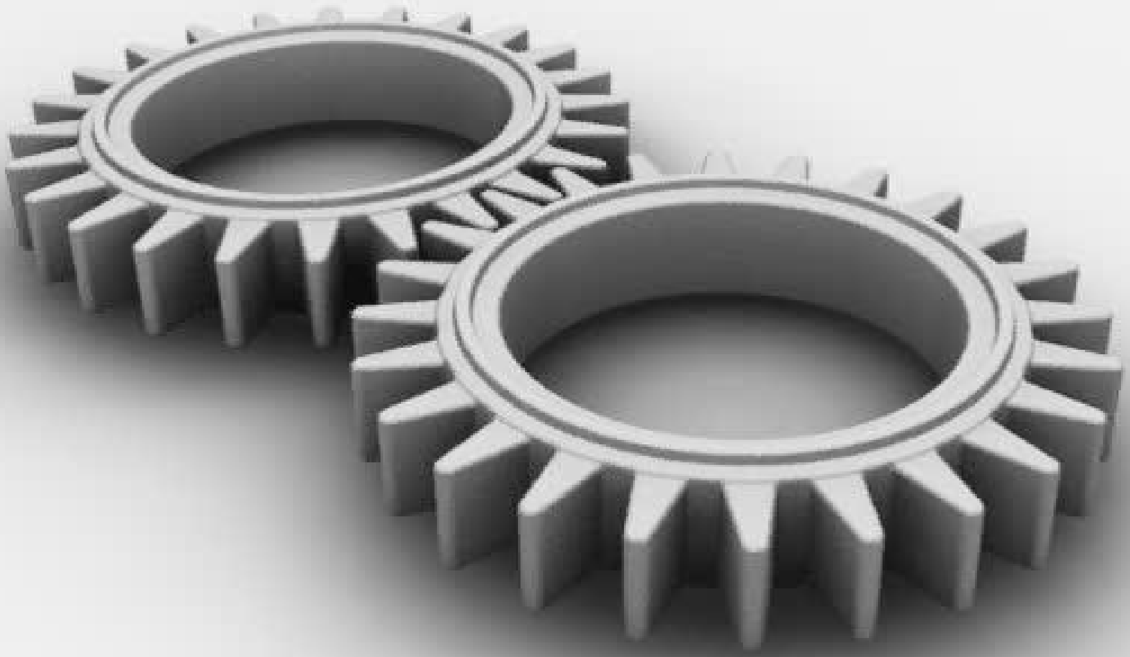
What makes code hard to unit test?

- Mutable Data
  - Generally things changing unpredictably

- Side Effects
  - Methods that modify things (vs. calculate, query)

- Dependencies (coupling vs. cohesion)
  - Singletons and concrete classes vs interfaces

How do you test this?



Start with testing this.

Unit Testing (Not just for finding bugs)

- Finds problems early
  - Makes us think of edge cases
- Facilitates change
  - Makes refactoring easier
- Simplifies integration
  - Small, testable components
- Self Documentation
  - Tests show how to use API
- Living Formal Design
  - Explicitly reveals dependencies

Writing code that’s easy to test…
is actually just **well written code**!

When to Unit Test

What to unit test:

- Risky code (IAP, tutorial, cause crashes, etc.)
- Frequently run code (hot spots, core loop classes)
- Shared base-code (regression testing)

Functional Programming

Provides:

● Immutable Data	Less mutable state == easier to test
● No Side Effects	Fewer moving parts == easier to test
● Composable	Re-usable pieces == easier to test
● Fewer Dependencies	More abstraction == easier to test

In OO:

- A method should do one thing well
- Single responsibility principle

Functional Programming

It’s not theoretical, anymore.

Multithreaded programming in the 90s:

- Mutexes
- Semaphores
- Spin-Locks

If you don’t use these correctly you have race-conditions and crashes. Over-use and you create bottlenecks and lose any multi-threading performance gains.

Functional Programming

The new way: blocks (closures) and run loops on threads

```
dispatch_async(myQueue, ^{
    // Perform long running process

    dispatch_async(dispatch_get_main_queue(), ^{
        // Update the UI
    });
});
```

This is a much easier to understand, modify and scale.
It’s roots are in functional programming - closures, no shared data

Example: Difficult to Test

```
class Foo {

    void DoSomething() {
        this.foo = Bar.baz += 1;
    }

    void Update() {
        DoSomething();
    }

}
```

The method DoSomething() is difficult to test because:

* It has side effects (modifies this.foo)
* It has external dependencies (singleton)

Example: Easy to Test

```
class Foo {

    private IBar bar;

    int DoSomething(int arg) {
        return this.bar.baz() + arg;
    }

    void Update() {
        this.foo = DoSomething(1);
    }

}
```

Now DoSomething(): * Has fewer dependencies
* Is modular / composable
* Has zero side effects

Terminology

- Continuous Integration
- Stub Methods
- Mock Objects
- Regression Testing
- Test-Driven Development
- Code Isolation

Functional programming

- Referential transparency

Promote Unit Testing

How to promote well-factored code:

- Design Previews
- Code Reviews
- Pair Programming
- Documentation
- Sprint Review

Multithreaded programming

Dependency Injection

- Constructor
  ```
  new Foo(Bar.X);
  ```
- Property
  ```
  foo.x = Bar.X;
  ```
- Method
  ```
  foo.qux(Bar.X);
  ```

```
Dani P - client side DI Unity

James T - server DI Unity?
```