Writing Testable Code

Bob Fornal

Entrepreneur

Code-squid provides solid, in-depth frontend training that is supported with real-world code projects. Blessed husband and proud father of two.

Senior Solutions Developer
Leading EDJE, Inc.

Passionate about learning, testing, mentoring, speaking, and personal growth.

Writing Testable Code

Embrace TESTABILITY ...

- Not just catching bugs.
- Foster ***a culture of quality and efficiency.***
- How easily software in a system can be tested.

Highly testable code allows for ...

- More efficient and effective identification of defects.
- Ensuring higher quality and reliability.

Key characteristics of testable code include ...

- **Modularity**: Code is organized into discrete units.
- **Clarity**: Code is understandable and its purpose is clear.
- **Independence**: Units of code can be tested in isolation without reliance on external systems or states.

Major Talk Sections

- Constructor does Real Work
- Collaborators
- Brittle Global State and Singletons
- Class does Too Much

Constructor does Real Work

Test setup is complicated by ***work in the constructor*** such as:

- Creating or initializing collaborators.
- Communicating with other services.
- Logic to set up its own state.

In the constructor in attribute declaration watch for ...

- `new` keyword.
- Static method calls.
- Control flow (conditional or looping logic).
- ... anything more than assignment.

When constructing the class in isolation or with test-double collaborators ...

- How hard is it?

If it's hard:

- You are doing too much work in the constructor!

If it's easy:

- Pat yourself on the back.

Collaborators

Avoid ***holder***, ***context***, and ***"kitchen sink"*** objects; they...

- Take all sorts of other objects.
- Are a grab bag of collaborators.
- Pass in the specific object you need as a parameter.
- Need to reach into one object, to get another, etc.

The chain does not need to be deep.

- If you count more than one period (.) in the chain, you're looking right at an example of this flaw.

Issues

- Objects are passed in but never used directly (only used to get access to other objects).
- Law of Demeter violation: method call chain walks an object graph with more than one dot (.).
- Suspicious names: context, environment, principal, container, or manager.

Brittle Global State and Singletons

Global State and Singletons obscure the true dependencies.

- Accessing global state statically does not clarify shared dependencies to readers of the code that uses the Global State.
- To really understand the dependencies, developers must read every line of code.

It causes ***Spooky Action at a Distance***:

- When running test suites, global state mutated in one test can cause a subsequent or parallel test to fail unexpectedly.

Break the static dependency using dependency injection.

Issues

- Adding or using singletons.
- Adding or using static fields or static methods.
- Adding or using static initialization blocks.
- Adding or using registries.
- Adding or using service locators.

NOTE:

> "Singleton" here is the classic Gang of Four singleton.
>
> An “application singleton” on the other hand is an object which has a single instance in our application, but which does not enforce its own "singletonness."

Spooky Action at a Distance

This can only happen via Global State.

This is when ...

- Code is run in isolation (nothing passed in).
- Unexpected interactions and state changes happen in distant locations of the system.

Use static state (Global State) ...

- Creates hidden communication channels.
- Obscures the codebase.

Spooky Action at a Distance ...

- Forces developers to read every line of code to understand the potential interactions.
- Lowers developer productivity.
- Confuses new team members.

When is Global State OK?

When the reference is a constant and the object it points to is either primitive or immutable.

```
const URL = "http://google.com";
```

... is OK since there is no way to mutate the value.

When the information only travels one way.

For example a Logger is one big singleton.

- However our application only writes to logger and never reads from it.
- More importantly our application does not behave differently based on what is or is not enabled in our logger.

Class Does Too Much

The class has too many responsibilities.

- Interactions between responsibilities are buried within the class.
- Tests do not have a clear seam between these interactions.

Construction of dependent components is a responsibility that should be isolated from the class's real responsibility.

- Use dependency injection to pass in pre-configured objects.
- Extract classes with single responsibilities.

Considerations ...

- Summing up what the class does includes the word and.
- Class would be challenging for new team members to read and "get it quickly."
- Class has fields that are only used in some methods.
- Class has static methods that only operate on parameters.

Class Does Too Much

When classes have a large span of responsibilities and activities, you end up with code that is:

- Hard to debug.
- Hard to test.
- Non-extensible system.
- Difficult for onboarding developers.
- Hard to hand off.
- Not subject to altering behavior via standard mechanisms: decorator, strategy, subclassing.
- A class that is hard to name.

Class Does Too Much (Fixing)

A class that does too much, should be split up ...

1. Identify the individual responsibilities.
2. Name each one crisply.
3. Extract functionality into a separate class for each responsibility.
4. One class may perform the hidden responsibility of mediating between the others.
5. Celebrate that now you can test each class in isolation much easier than before.

If working with a legacy class that did too much, and you can't fix the whole legacy problem today, you can at least:

1. Sprout a new class with the sole responsibility of the new functionality.
2. Extract a class where you are altering existing behavior.

Writing Testable Code

Testability is ...

- Not a standalone feature.
- It is a fundamental aspect of ***good code design***.

To develop more robust and maintainable code, developers need to (from the start) consider:

- End-user experience.
- System requirements.

Ensuring that the code not only meets its ...

- ***Functional requirements***.
- ***Resilient and adaptable to change***.

We've covered ...

- Constructor does Real Work
- Collaborators
- Brittle Global State and Singletons
- Class does Too Much

TIME FOR QUESTIONS