# Unit Testing Code with Hard-to-Mock Dependencies

Adam Wolfe Gordon
DigitalOcean

## What Are Mocks?

- Martin Fowler’s definition*:
    - Objects pre-programmed with expectations which form a specification of the calls they are expected to receive.
- “Mocks” in this talk means “test doubles.”

* https://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs

GOLAB

## Why Use Mocks?

- Decouple tests from the real world.
- Allow for testing without running dependencies.
- Force hard-to-simulate error conditions.

# Basic Example: Without Mocks

```go
package ip

// GetIP returns my external IP address.
func GetIP() (net.IP, error) {
	url := "https://icanhazip.com"
	resp, err := http.Get(url)
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()

	content, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		return nil, err
	}

	str := strings.TrimSpace(string(content))
	ip := net.ParseIP(str)
	if ip == nil {
		return nil, errors.New("invalid IP")
	}
	return ip, nil
}
```

```go
package ip_test

func TestGetSuccess(t *testing.T) {
	ip, err := ip.GetIP()
	assert.NoError(t, err)
	assert.NotNil(t, ip)
}
```

What if we can't reach icanhazip.com?

Would be nice to make a stronger assertion

Tests use github.com/stretchr/testify/assert

# Basic Example: With Mocks

```go
package ip

// GetIP returns my external IP address.
func GetIP(getter HTTPGetter) (net.IP, error) {
	url := "https://icanhazip.com"
	resp, err := getter.Get(url)
	if err != nil {
		return nil, err
	}
	defer resp.Body.Close()

	content, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		return nil, err
	}

	str := strings.TrimSpace(string(content))
	ip := net.ParseIP(str)
	if ip == nil {
		return nil, errors.New("invalid IP")
	}
	return ip, nil
}
```

```go
type HTTPGetter interface {
	Get(url string) (*http.Response, error)
}
```

```go
package ip_test

func TestGetSuccess(t *testing.T) {
	respBody := &noopReadCloser{
		bytes.NewBufferString("127.0.0.1\n"),
	}
	resp := &http.Response{
		Body: respBody,
	}

	getter := &ip.MockHTTPGetter{}
	getter.On("Get", "https://icanhazip.com").
		Return(resp, nil).
		Once()

	ip, err := ip.GetIP(getter)
	assert.NoError(t, err)
	assert.Equal(t, net.IPv4(127, 0, 0, 1), ip)
}
```

Mocks generated using github.com/vektra/mockery, which uses github.com/stretchr/testify/mock

## Easy-to-Mock Dependencies

- Export interfaces or structs with methods.
- Have accessor methods for data.
- Don’t use cgo.
- For example: `net/http`.

## Hard-to-Mock Dependencies

- Have free functions (not methods).
- Have functions that return complex structs.
- Use cgo.
- For example: os.

## Techniques

## Technique 1
## Isolation

- Factor out code that doesn’t depend on your hard-to-mock dependency.
- Write tests for the factored out functions.
- Main control flow is a series of calls to hard-to-mock things and well-tested functions.

## Example: File Server

```go
func serveFile(w http.ResponseWriter, r *http.Request) {
	path := r.URL.Path
	if path == "" || path == "/" {
		w.WriteHeader(http.StatusBadRequest)
		return
	}
	path = filepath.Join("/tmp", path)
	st, err := os.Stat(path)
	switch {
	case os.IsNotExist(err):
		w.WriteHeader(http.StatusNotFound)
		return
	case os.IsPermission(err):
		w.WriteHeader(http.StatusForbidden)
		return
	case err != nil:
		w.WriteHeader(http.StatusInternalServerError)
		return
	case st.IsDir():
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	f, err := os.Open(path)
	if err != nil {
		w.WriteHeader(http.StatusInternalServerError)
		return
	}
	defer f.Close()

	w.WriteHeader(http.StatusOK)
	io.Copy(w, f)
}
```

# Example: File Server, With Isolation

```go
func serveFile(w http.ResponseWriter, r *http.Request) {
	path, err := getPath(r)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	st, err := os.Stat(path)
	code := statOK(st, err)
	if code != http.StatusOK {
		w.WriteHeader(code)
		return
	}

	f, err := os.Open(path)
	if err != nil {
		w.WriteHeader(http.StatusInternalServerError)
		return
	}
	defer f.Close()

	respondSuccess(w, f)
}
```

```go
func getPath(r *http.Request) (string, error) {
	path := r.URL.Path
	if path == "" || path == "/" {
		return "", errors.New("no path in request")
	}
	return filepath.Join("/tmp", path), nil
}

func statOK(st os.FileInfo, err error) int {
	switch {
	case os.IsNotExist(err):
		return http.StatusNotFound
	case os.IsPermission(err):
		return http.StatusForbidden
	case err != nil:
		return http.StatusInternalServerError
	case st.IsDir():
		return http.StatusBadRequest
	}

	return http.StatusOK
}

func respondSuccess(w http.ResponseWriter, f io.Reader) {
	w.WriteHeader(http.StatusOK)
	io.Copy(w, f)
}
```

# Example: File Server, Tests With Isolation

```go
func TestGetPathSuccess(t *testing.T) {
	u, _ := url.Parse("http://localhost:8080/foo.go")
	req := &http.Request{
		URL: u,
	}

	path, err := getPath(req)

	assert.Equal(t, "/tmp/foo.go", path)
	assert.NoError(t, err)
}

func TestGetPathEmpty(t *testing.T) {
	u, _ := url.Parse("http://localhost:8080/")
	req := &http.Request{
		URL: u,
	}

	_, err := getPath(req)

	assert.Error(t, err)
}
```

```go
func TestStatOK(t *testing.T) {
	tcs := []struct {
		name         string
		st           os.FileInfo
		err          error
		expectedCode int
	}{
		{
			name:         "not exist",
			st:           nil,
			err:          os.ErrNotExist,
			expectedCode: http.StatusNotFound,
		},
		{
			name:         "permission",
			st:           nil,
			err:          os.ErrPermission,
			expectedCode: http.StatusForbidden,
		},
		...
	}

	for _, tc := range tcs {
		t.Run(tc.name, func(t *testing.T) {
			code := statOK(tc.st, tc.err)
			assert.Equal(t, tc.expectedCode, code)
		})
	}
}
```

## Technique 2
## Wrapping

- Wrap hard-to-mock code in local interfaces and structs.
- Allows for dependency injection.
- Allows for standard mocking techniques.
- Adds a level of indirection.
- Requires some extra code in production just to allow for testing.

## How to Wrap a Dependency

1. Create interfaces that match the dependency’s function signatures.
2. Create a struct that implements the interface by passing calls through to the dependency.
3. Replace return types with interfaces where possible.
4. Write more complex wrappers for return types if needed.

## Example: File Server, With Wrapping

```go
type OS interface {
	Stat(path string) (os.FileInfo, error)
	Open(path string) (io.ReadCloser, error)
}

type realOS struct{}

func (*realOS) Stat(path string) (os.FileInfo, error) {
	return os.Stat(path)
}

func (*realOS) Open(path string) (io.ReadCloser, error) {
	return os.Open(path)
}
```

```go
type handler struct {
	os OS
}

func (h *handler) serveFile(w http.ResponseWriter,
                            r *http.Request) {
	path := r.URL.Path
	if path == "" || path == "/" {
		w.WriteHeader(http.StatusBadRequest)
		return
	}
	path = filepath.Join("/tmp", path)

	st, err := h.os.Stat(path)
	if code := statOK(st, err); code != http.StatusOK {
		w.WriteHeader(code)
		return
	}

	f, err := h.os.Open(path)
	if err != nil {
		w.WriteHeader(http.StatusInternalServerError)
		return
	}
	defer f.Close()

	w.WriteHeader(http.StatusOK)
	io.Copy(w, f)
}
```

# Example: File Server, Tests With Wrapping

```
func TestServeFileSuccess(t *testing.T) {
        mos := &MockOS{}
        h := &handler{
                os: mos,
        }

        mos.On("Stat", "/tmp/foo.go").
                Return(&fakeStat{false}, nil).
                Once()
        mos.On("Open", "/tmp/foo.go").
                Return(&noopReadCloser{
                        bytes.NewBufferString("hello"),
                }, nil).
                Once()

        w := httptest.NewRecorder()
        u, _ := url.Parse("http://localhost:8080/foo.go")
        req := &http.Request{
                URL: u,
        }

        h.serveFile(w, req)

        assert.Equal(t, http.StatusOK, w.Code)
        assert.Equal(t, "hello", w.Body.String())
}
```

```
func TestServeFileOpenError(t *testing.T) {
        mos := &MockOS{}
        h := &handler{
                os: mos,
        }

        mos.On("Stat", "/tmp/foo.go").
                Return(&fakeStat{false}, nil).
                Once()
        mos.On("Open", "/tmp/foo.go").
                Return(nil, errors.New("oops!")).
                Once()

        w := httptest.NewRecorder()
        u, _ := url.Parse("http://localhost:8080/foo.go")
        req := &http.Request{
                URL: u,
        }

        h.serveFile(w, req)

        assert.Equal(t, http.StatusInternalServerError, w.Code)
}
```

## Diversion: Mocking in C

- C doesn’t have classes or structs with methods: all functions are free functions.
- In C we mock using linker tricks: build a fake version of a dependency, and link your tests against it instead of the real version.
- Could we do the same in Go?

## Introducing mockpkg

- mockery and similar tools can only mock interfaces - hence the wrapper approach.
- mockpkg is a tool I wrote to mock free functions.
- mockpkg generates an interface from a package's free functions, then generates a mock for that interface.

mockpkg is available at https://github.com/adamwg/mockpkg.

## Technique 3
## Package Mocking

- Create a variable for the free functions you’re using from your dependency.
- Use mockpkg to generate a mock.
- In tests, create a mock object and assign the function variables.
- Allows for tests similar to wrapping, but without the overhead of manual wrapping.

## Example: Time Server

```go
func serveTime(w http.ResponseWriter, r *http.Request) {
	path := r.URL.Path
	if path == "" || path == "/" {
		serveNow(w)
	} else {
		serveSince(w, path[1:])
	}
}

func serveNow(w http.ResponseWriter) {
	t := time.Now().Format(time.RFC3339) + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(t))
}

func serveSince(w http.ResponseWriter, from string) {
	t, err := time.Parse(time.RFC3339, from)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	d := time.Since(t).String() + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(d))
}
```

# Example: Time Server, With mockpkg

```go
var (
	now       = time.Now
	since     = time.Since
	parseTime = time.Parse
)

func serveNow(w http.ResponseWriter) {
	t := now().Format(time.RFC3339) + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(t))
}

func serveSince(w http.ResponseWriter, from string) {
	t, err := parseTime(time.RFC3339, from)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	d := since(t).String() + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(d))
}
```

```go
var (
	mockTime      = &mocks.Time{}
)

func init() {
	now = mockTime.Now
	since = mockTime.Since
	parseTime = mockTime.Parse
}
```

# Example: Time Server, With mockpkg (ctd.)

```go
var (
	now       = time.Now
	since     = time.Since
	parseTime = time.Parse
)

func serveNow(w http.ResponseWriter) {
	t := now().Format(time.RFC3339) + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(t))
}

func serveSince(w http.ResponseWriter, from string) {
	t, err := parseTime(time.RFC3339, from)
	if err != nil {
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	d := since(t).String() + "\n"
	w.WriteHeader(http.StatusOK)
	w.Write([]byte(d))
}
```

```go
var (
	fakeNow       = time.Unix(1136239445, 0)
	fakeNowString = fakeNow.Format(time.RFC3339) + "\n"
)

func TestServeNowSuccess(t *testing.T) {
	mockTime.On("Now").Return(fakeNow).Once()

	w := httptest.NewRecorder()
	serveNow(w)

	assert.Equal(t, http.StatusOK, w.Code)
	assert.Equal(t, fakeNowString, w.Body.String())
}

func TestServeSinceSuccess(t *testing.T) {
	arbitrary:= time.Unix(1234567890, 0)
	s := arbitrary.Sub(fakeNow)
	expectedOut := s.String() + "\n"

	mockTime.On("Parse", time.RFC3339, fakeNowString).
		Return(fakeNow, nil).Once()
	mockTime.On("Since", fakeNow).Return(s).Once()

	w := httptest.NewRecorder()
	serveSince(w, fakeNowString)

	assert.Equal(t, http.StatusOK, w.Code)
	assert.Equal(t, expectedOut, w.Body.String())
}
```

## Technique 4
## Combining the Other Techniques

- Non-trivial codebases will need combinations of the techniques we've discussed.
- Wrap dependencies in a separate package.
- Isolate the calls to dependencies within that package.
- Use mockpkg to mock out the entire helper package.

## Example: Volume Formatting Server

```go
package rpcserver

type rpcServer struct {
	...

	// The following functions are configurable for testing purposes.

	// hasFilesystem returns true if the device at a given path has been
	// formatted.
	hasFilesystem func(ctx context.Context, devicePath string) (bool, error)
	// mkfs creates a filesystem of the given type on the given device.
	mkfs func(ctx context.Context, fs string, path string, label string) error
	// formatInfo gets info about the format of the given device.
	formatInfo func(ctx context.Context, path string) (*fs.VolumeFormatInfo, error)
}
```

## Example: Volume Formatting Server (ctd.)

```go
func Mkfs(ctx context.Context, fs string, path string, label string) error {
	cmd := mkfsCommand(fs, path, label)
	rc, err := run(ctx, cmd[0], cmd[1:]...)
	if err != nil {
		span.SetError(err)
		return err
	}


	if rc != 0 {
		return fmt.Errorf("formatting failed with code %d", rc)
	}

	return nil
}

func Info(ctx context.Context, devicePath string) (*VolumeFormatInfo, error) {
	cmd := exec.CommandContext(ctx, "blkid", "-p", "-o", "export", devicePath)
	rc := 0
	out, err := cmd.Output()
	if err != nil {
		if exitError, ok := err.(*exec.ExitError); ok {
			ws := exitError.Sys().(syscall.WaitStatus)
			rc = ws.ExitStatus()
		} else {
			return nil, err
		}
	}

	return parseBlkidOutput(out, rc)
}
```

## Conclusions

## Isolation: Pros and Cons

### Pros

- Low overhead.
- Allows for decent coverage.
- May improve code structure.

### Cons

- Leaves calls to dependencies untested.
- Structure may be unnatural.

## Wrapping: Pros and Cons

### Pros

- Allows for excellent coverage.
- Uses standard techniques.
- May improve code structure.

### Cons

- Adds some indirection that may make code non-obvious.
- Possible performance hit.
- Wrapper structures can’t be easily tested, could have bugs.
- Structure may be unnatural.

# Package Mocking: Pros and Cons

**Pros**

- Allows for excellent coverage.
- Retains code structure.
- Limited added indirection.
- Limited added code.

**Cons**

- Somewhat non-standard.
- Possible performance hit.
- Function variables can be set incorrectly, leading to bugs.
- Concurrency in tests is tricky.

Thank You!

Adam Wolfe Gordon

awg@do.co

DigitalOcean

GOLAB

DigitalOcean