Testing Strategies for a Go Application

James Culbertson
Golang Houston Meetup
January 21, 2016
Outline

● Why Do Testing?
● About the Application
● Testing Challenges and Strategies Used
● Recommendations and Links
Why Do Testing?

- Refactor With More Confidence
  - Cleanup and improve existing code
  - Release updates quickly and more confidently
- Add New Features With Fewer Regressions
- Development is Enhanced by Testing
  - Encourages decoupled designs via interfaces
  - Encourages separation of concerns
- Go Makes it Very Easy
  - Nice built-in testing platform
  - Helps do package development without main()
Why Do Testing?

- Other Benefits
  - Documents usage of public methods
  - Gives more confidence/acceptance to developers who will use your open source code

- Not a “Silver Bullet”
  - Requires thought, effort, design, and practice
  - Code coverage is a metric but not the end goal
About the Subject Application

- Developed New Application Go Ping Sites
  - To better learn Go
  - To fulfill a need

- Application Monitors Availability for Websites
  - Background goroutines to ping websites
  - Notifies users of change in status via Email and SMS
  - Includes website for status, reports, and settings
  - Local sqlite3 database to store sites, users, and history

- Good Application to Explore Testing in Go
Application Components

Websites

Pings at site-configured intervals

Go Ping Sites App

Pinger

Database

Notifier

Web

DB

Text, Email Notifications

Status, Reports, Settings
## Monitored Sites Status

<table>
<thead>
<tr>
<th>Website</th>
<th>Status</th>
<th>Since</th>
<th>Last Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.com</td>
<td>Up</td>
<td>Unknown</td>
<td>6 seconds ago</td>
</tr>
<tr>
<td>example.org</td>
<td>Up</td>
<td>Unknown</td>
<td>6 seconds ago</td>
</tr>
</tbody>
</table>

*Page refreshed: 2016-01-09 12:27:36 PM*
Testing Mechanics

- To Test All Packages and Methods
  
  `go test ./...`

- To Test All Methods in a Specific Package
  
  `go test -v ./packagename`

- To Test Specific Methods in a Package
  
  `go test -v ./packagename -run name`
  
  - where `name` is a regular expression

- For example to test all notifier tests with Error in name
  
  `go test -v ./ notifier -run Error`
Testing Challenges

- Goroutines
  - “Pinger” and “Notifier” Components
  - How to “white-box” test public methods?
  - How to assert results?

- Method I Used
  - Test methods as package user via different package name
  - Log key events to log file
  - Import log file and ensure it contains expected results
  - Inject mocks when necessary for dependencies
Using Different Package Name

- Ensures some level of isolation ("white box" testing)
- Requires importing package and using public methods
Public Method Testing

- Using public methods from package
- This example injects mocks explicitly as arguments

```go
func TestStartPinger(t *testing.T) {
    // Fake db for testing.
    db, _ := sql.Open("testdb", ">
    pinger.CreatePingerLog(""
    p := pinger.NewPinger(db, pinger.GetSitesMock, pinger.RequestURLMock,
        pinger.DoExitMock, notifier.SendEmailMock, notifier.SendSmsMock)
    p.Start()
    time.Sleep(3 * time.Second)
    p.Stop()
}
```
Logging Key Events

- Logging within go routine:

```go
if !ls.IsActive {
    log.Println(s.Name, "Paused")
    pause(s.PingIntervalSeconds)
    continue
}
```

- Read log file inside test to verify results:

```go
results, err := pinger.GetLogContent()
if err != nil {
    t.Fatal("Failed to get log results.", err)
}
if !strings.Contains(results, "Test 3 Paused") {
    t.Errorf("Failed to report paused site.")
}
```
Mocking

When to Use
- Create output from third-party package or service
- Faster tests by mocking slower components such as DB access
- Setup error conditions to test error handling

One Solution
- Pass functions as values in struct or method args
- Easy to do without mocking library
Mocking

- Define a type for the function to be mocked

```go
// EmailSender defines a function to do the work to send an email.
type EmailSender func(recipient string, message string, subject string) error
```

- Provide function to initialize object to pass the mock

```go
func NewNotifier(site database.Site, message string, subject string,
    sendEmail EmailSender, sendSms SmsSender) *Notifier {
    n := Notifier{Site: site, Message: message, Subject: subject,
        SendEmail: sendEmail, SendSms: sendSms}
    return &n
}
```

- Define mock with same signature as type

```go
func SendEmailErrorMock(recipient string, message string, subject string) error {
    return errors.New("Error - no response from server.")
}
```
Mocking

- Inject mock into test functions

```golang
func TestNotifyError(t *testing.T) {
    site := getTestSite()
    n := notifier.NewNotifier(site, message, subject, notifier.SendEmailErrorMock,
    n.Notify()
}
```

- Create implementations to inject for production use

```golang
// Set up authentication information.
auth := smtp.PlainAuth("", config.Settings.SMTP.EmailAddress, config.Set
    config.Settings.SMTP.Server)
server := config.Settings.SMTP.Server + "" + config.Settings.SMTP.Port
```
Testing With Interfaces

- **When to Use**
  - Solving difficult issues around third-party packages
  - Clean way to implement and mock your own components

- **Example – apexskier/httpauth**
  - Third Party package providing authorization
  - Needed to mock “CurrentUser” and “Messages” methods for testing handlers/controllers
  - Key was to provide an interface for those methods and implement both mock and concrete methods
Testing With Interfaces

- Problem
  - Dependence on Methods Coupled to Web State

```go
type aboutController struct {
    template *template.Template
    authorizer httpauth.Authorizer
}

func (controller *aboutController) get(rw http.ResponseWriter, req *http.Request) {
    isAuthenticated, user := controller.authorizer.CurrentUser(rw, req)
    messages := controller.authorizer.Messages(rw, req)
    vm := viewmodels.GetAboutViewModel(isAuthenticated, user, messages)
    controller.template.Execute(rw, vm)
}
```

```go
    isAuthenticated = false
    var err error
    user, err = authorizer.CurrentUser(rw, req)
    if err != nil {
        // Handle error
    }
}
```
Testing With Interfaces

● Solution

● Declare an interface with the same method signatures

```go
type CurrentUserGetter interface {
    Messages(rw http.ResponseWriter, req *http.Request) [][]string
}
```

● Pass the reference to the interface type instead

```go
type aboutController struct {
    template  *template.Template
    authorizer CurrentUserGetter
}
```

● External library will still work as before
Testing With Interfaces

- Create struct for mocking the methods/passing properties

```go
type MockCurrentUserGetter struct {
    FlashMessages []string
    Username     string
    UserError    error
}
```

- Define the mock methods

```go
    user = httpauth.UserData{Username: m.Username}
    return user, m.UserError
}
```

```go
func (m MockCurrentUserGetter) Messages(rw http.ResponseWriter, req *http.Request) []string {
    messages := m.FlashMessages
    return messages
}
```
Testing With Interfaces

- In test, setup mock object and pass to controller to be used in handler

```go
func TestAboutController(t *testing.T) {
    mockUserGetter := MockCurrentUserGetter{Username: "jules",
        FlashMessages: []string{"Log in to do that."}}

    req, _ := http.NewRequest("GET", "/about", nil)
    templates := PopulateTemplates("../templates")
    ac := new(aboutController)
    ac.template = templates.Lookup("about.gohtml")
    ac.authorizer = mockUserGetter

    w := httptest.NewRecorder()
    ac.get(w, req)
}```
HTTP Testing

- Go Provides httptest Package
  - “Server” for creating mocks for testing client requests, for example to mock external services
  - “ResponseRecorder” for recording responses, for example to test the response of the handlers

```go
func (h *statusHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(int(*h))
}

func TestIsInternetAccessibleFirstSiteBad(t *testing.T) {
    // Set up a non-running server and a good server
    statusOK := statusHandler(http.StatusOK)
    s1 := httptest.NewUnstartedServer(&statusOK)
    s2 := httptest.NewServer(&statusOK)
    defer s1.Close()
    defer s2.Close()
}
```
Testing Databases

- **Database**
  - Usually mock DB when testing other components
  - Also important to have some actual DB tests
  - Each test should stand on its own (initialization)

- **One Solution for Testing DB Methods**
  - Build an initialize method that also creates schema
  - In each test clear DB and run initialization
  - Run DB access tests as user of package
  - Test pattern is insert/update then read/verify
  - Be sure to test conditions that should violate constraints
Database Initialization

- `InitializeDB` can create a new test DB and schema:

```go
func InitializeDB(dbPath string, seedFile string) (*sql.DB, error) {
    newDB := false
    if _, err := os.Stat(dbPath); os.IsNotExist(err) {
        newDB = true
    }

    db, err := sql.Open("sqlite3", dbPath)
    if err != nil {
        return nil, err
    }

    if newDB {
        fmt.Println("New Database, creating Schema...")
        err = createSchema(db)
        if err != nil {
            return nil, err
        }
    }
}
```
Database Initialization

- For testing use a wrapper to also delete the old DB:

```go
// InitializeTestDB is for test packages to initialize a DB for integration tests.
func InitializeTestDB(seedFile string) (*sql.DB, error) {
    var db *sql.DB
    err := deleteDb(testDb)
    if err != nil {
        return nil, err
    }
    db, err = InitializeDB(testDb, seedFile)
    if err != nil {
        return nil, err
    }
    return db, nil
}
```
Database Initialization

- For testing use a wrapper to also delete the old DB:

```go
// InitializeTestDB is for test packages to initialize a DB for integration testing.
func InitializeTestDB(seedFile string) (*sql.DB, error) {
    var db *sql.DB
    err := deleteDb(testDb)
    if err != nil {
        return nil, err
    }
    db, err = InitializeDB(testDb, seedFile)
    if err != nil {
        return nil, err
    }
    return db, nil
}
```
Database Testing

- Example of create, get, and verification

```go
err = c2.AddContactToSite(db, site.SiteID)
if err != nil {
    t.Error("Failed to associate contact2 with site:", err)
}

// Get the saved site contacts
err = site.GetSiteContacts(db, site.SiteID)
if err != nil {
    t.Error("Failed to retrieve site contacts:", err)
}

// Verify the second contact was loaded as the first in list by sort order
if !reflect.DeepEqual(c2, site.Cntacts[0]) {
    t.Error("New contact saved not equal to input:
", site.contacts[0], c2)
}
```

Caution: Have seen issues with DeepEqual comparisons with zero time objects.
Test Coverage

- Coverage testing of a package
  
  ```go
go test ./packagename -cover
  ```

- Graphical coverage
  
  ```go
  go test ./packagename -coverprofile=coverage.out
  go tool cover -html=coverage.out
  ```

```go
// isInternetAccessible checks two highly available sites to check whether the
// outside Internet is responding and there are no internal network problems.
func isInternetAccessible() bool {
  to := time.Duration(5) * time.Second
  client := http.Client{
    Timeout: to,
  }
  _, err1 := client.Get("http://www.example.com")
  if err1 != nil {
    if err2 != nil {
      return false
    }
  }
  return true
}
```
Test Coverage

- Coverage testing of a package
  
  ```go
go test ./packagename -cover
  ```

- Graphical coverage
  
  ```go
go test ./packagename -coverprofile=coverage.out
  go tool cover -html=coverage.out
  ```
Test Coverage

- Heat Maps Indicate Intensity of Coverage

```
go test -covermode=count ./<packagename>
-coverprofile=count.out

-go tool cover -html=count.out
```
Recommendations

- Use Go's Native Testing Environment
- Write Tests During Development
- Separate Functional and Unit Tests
  - Prefer functional tests - less brittle to refactoring changes than unit tests
  - Use different package name in functional tests to enforce public only
  - Test private functions when required in separate test file with same package as one under test
Recommendations

- When a Defect is Found – Write a Test To Cover
- Test Your Code Behavior Not Vendor Packages
  - Exception if some vendor packages need more tests
- Interfaces Are Your Friend
  - Allows mocking third-party methods and keeps things loosely coupled
- Use the Graphical Coverage Tools
  - Find important areas that need coverage
  - Don't worry about less than 100% coverage
Some Links

● Interfaces and Composition for Effective Unit Testing in Golang
  ● http://goo.gl/4jV5XD

● 5 simple tips and tricks for writing unit tests in #golang
  ● https://goo.gl/uuOBJj

● “Go Ping Sites” Discussed Today (under development)
  ● https://github.com/turnkey-commerce/go-ping-sites