Unit Testing from Soup to Nuts

Philippe Casgrain
Lead Developer
Lightspeed

Warning

- This presentation contains live coding. Anything that can go wrong, will go wrong (Murphy *et al*., 1952).
- Security is barely mentioned
- You will have access to all the code that is part of this talk.

Don’t put your private source code on a public repository. Use ssh keys and other best practices.
Don’t put your CI server on the public internet.

In this talk:

- Installing the XojoUnit framework
- Running Unit Tests in your app
- Running Unit Tests from the command-line
- Setting up a CI server
- Using JenkinsCI
- Factoring your Xojo project for Unit Testing

Cross-platform

I'm doing this on Mac, but you should be able to do the same on Linux or Windows

The software I'm using is cross-platform, and can be installed on multiple operating systems.

Getting started

We'll use the Sample Application « ToDoDesktop » and call it TaskMaster.

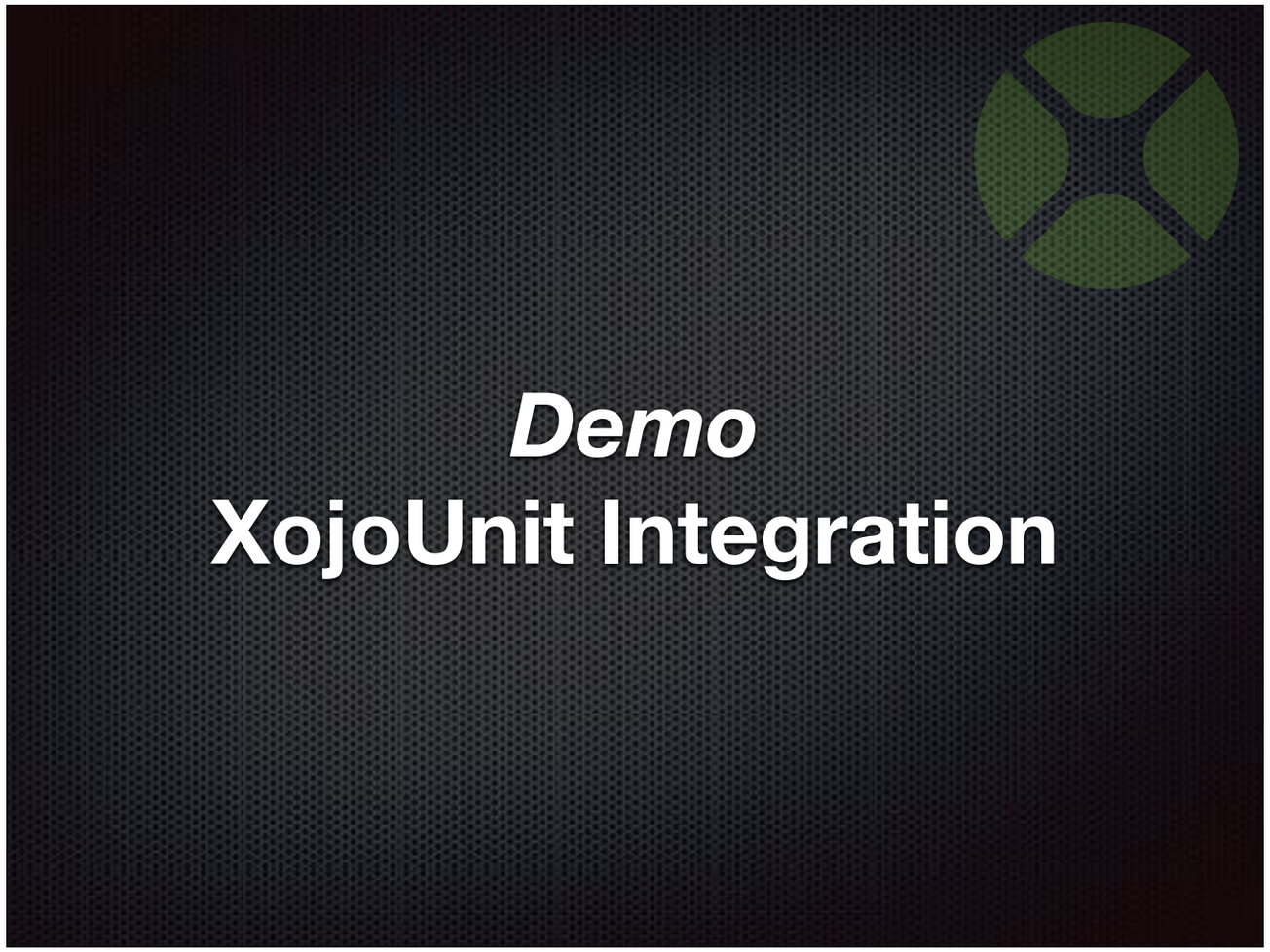
We'll put it under version control.

Create a project using the example Sample Applications > ToDoDesktop
Enter the appropriate template information:
Mac App Name: TaskMaster
Arch: 32 bit (since we want to run, too)
Bundle Identifier: com.lightspeedhq.taskmaster
Save as: Xojo Project, TaskMaster
Run app
Check resizing behaviour and fix it
Go to terminal: git init; git add *; git commit -a -m "Initial version"
Add to github: TaskMaster

Integrating XojoUnit

Xojo's official unit test framework

https://github.com/xojo/XojoUnit

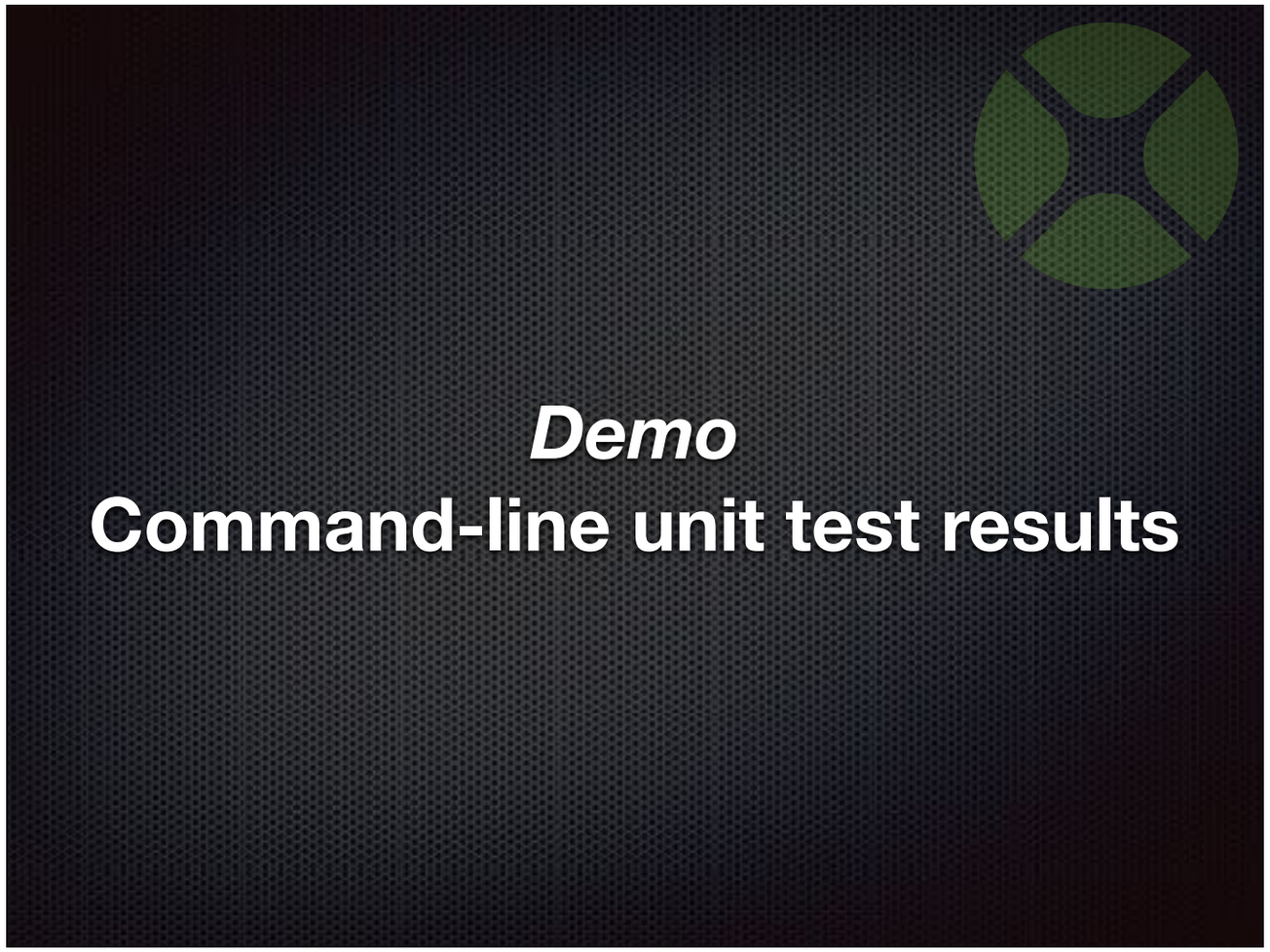


Clone from https://github.com/xojo/XojoUnit
Locate Desktop Project / XojoUnitDesktop.xojo_project and open
Run to show sample project
Run all tests
Locate the one failure and fix it
Run again to show all tests passing
Drag the whole XojoUnit folder to my app
Save, run -> Nothing
Add this to a new App > Open Event Handler: XojoUnitTestWindow.Show()
Run app, show tests
Export results
Commit changes

Command-line testing

Running the tests manually is fine for development.

What you really want is to automate gathering those results.

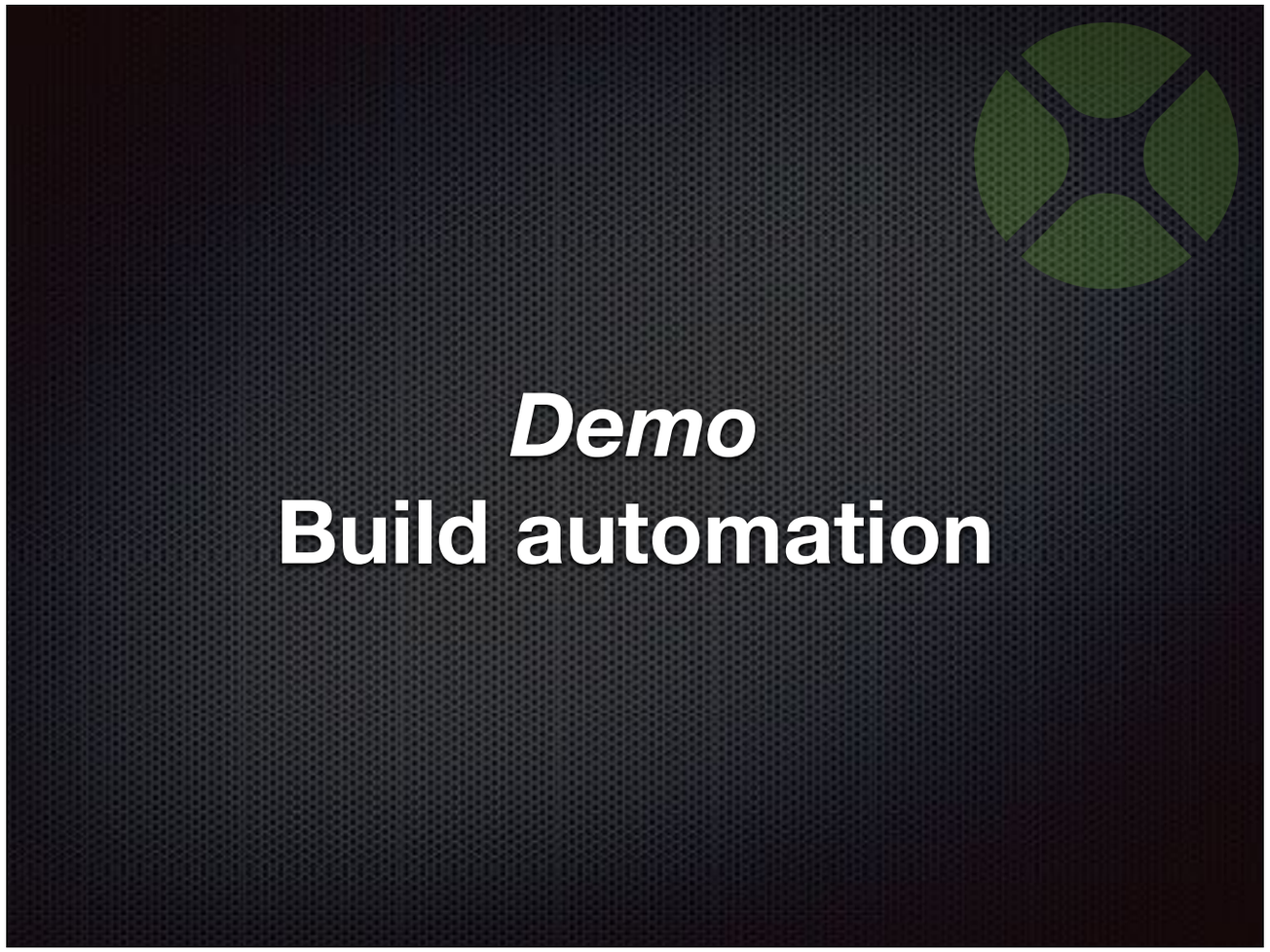


Build application, show it in the Finder
Uses Build Folder - no
Double-click to launch
Run tests - export results
Quit
Drag app in Terminal
Launch from command-line - run tests
Launch from command-line with --runUnitTests /Users/phil/Desktop/results.xml
Examine results file
Commit changes

Automating the build

- We want to repeat the build / test process without human intervention
- To build, we'll send XojoScript commands to the IDE
- To test, we'll use our command-line interface
- This example uses bash, it can be adapted for your shell

There are two steps: build, then test.
You could only do build if you were running a build machine, or doing an optimized Release build.



Locate both scripts: build.sh and unit-test.sh
Copy them in the project folder
Go over them
Run them
Add to version control and commit

Continuous Integration

- Running your unit tests in an automated way
- Also used to create build artifacts
  - Useful to point your users to a release folder
- Removes human intervention from the build process

CI Servers

- Bamboo
- Jenkins
- TeamCity
- …
- https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software

Most require Java (JDK 8 Update 101 as I write this)

Installing Jenkins

- https://jenkins.io
- Select your installer or download a .war file
- Run with
  `java -jar jenkins.war`
- Install command-line tools to get git support

Setup: Mac VM with Xojo Install
Xojo install needs a license; in my case I can license up to 3 computers because Pro
Xcode command-line tools already installed, for git
Java already installed
Go to Jenkins.io and download jenkins.war
java -jar jenkins.war
http://localhost:8080 and setup
Install only git plugin
Switch to the non-Vm browser
Create freestyle project, set to poll every minute
Run project

Success!

Extra credit

Refactoring for Testability

Factoring your project for testability

TaskMaster has no real unit tests.

Let's remove the boilerplate tests and add our own.

A quick primer on MVC

- Model: stores the data
- View: displays the data
  - Also allows user to update the data
- Controller: glues the two together

Model

- Highly unit-testable
- Highly re-usable
  - The same model layer (and tests) can often be used across architectures (32 vs 64 bits, ARM vs Intel, …)

View

- Rarely unit-testable
- Usually re-usable

Controller

- Somewhat testable
  - Lots of associated object
  - Mock objects
- Rarely re-usable

ListBox

- A model
- A view
- A controller
- Not easily testable

Here we switch to the project and inspect its code
Notice how everything ties into the ListBox
We'll call the ListBox our view, since it handles events really well and displays things, too

New classes

- ToDoItem
- ToDoList
- ToDoController

Let's start refactoring the sample application.
First, delete the sample tests
For more on refactoring, stick around for the next talk "Adventures in Refactoring"

ToDoItem

- Simple representation with a Bool and a String
- Knows how to write itself to a string, and read itself from a string

Create the ToDoItem class and its associated Test class
Save, commit and push, the results
We'll see if we broke anything

ToDoList

- Collection of ToDoItem
- Knows how to write itself to a string

Write the class quickly, foregoing the tests for time reasons
Save, commit and push

ToDoController

- Has a ListBox
- Has a ToDoList
- Handles button events

Q & A

Philippe Casgrain

philippe.casgrain@lightspeedhq.com

Give us feedback on this session in the XDC app!

Resources

- https://github.com/philippec-ls/TaskMaster
- https://github.com/xojo/XojoUnit