An Automated Mock Object Generator for C++

Scott Miller

Greg Pattison

Welcome!

Who We Are:

- Scott Miller & Greg Pattison of Atomic Object
- 25 years (combined) of software development
- Employing Agile techniques since 2005

Why?

Why we wanted a C++ mock generator:

- Began working together on a new C++ project
- We wanted to apply interactive-based testing techniques we had used on previous projects (.Net & Java)
- Tried hand-coding a few mocks, but this was time intensive and error prone

Evolution

- We looked around for an existing product or project.
- We found some stuff (mockcpp for one), but not a "Record & Playback" verification system we liked so much for our Java and C# projects

Evolution

- We wrote a few Python scripts to generate templates to ease the hand-coding of mocks

Evolution

- The discovery of GCCXML led to the first full-blown mock object code generator.
- The generator evolved over the course of the project, gaining functionality, but also acquiring a QT dependency

Evolution

- Moxy 2.0 is what we are showing you today. It has no platform dependencies

GCCXML

The Greatest Invention Known to Man

- Before this project had begun, I had played around a couple times with parsing C++ header files, but never really got anywhere.
- Then, early in the C++ project I read an article that mentioned a cool project that parsed C++ code into XML.

GCCXML

- What’s more, there was also a Python library that read in the XML result of the parse and presented the information in a simple class.
- This was the tool we were waiting for. It opened up the possibility of generating C++ code based on an existing interface definition.

GCCXML

- Thanks to Brad King, Kitware developers and The Insight Consortium for this fabulous tool.

The Interface

An example of a C++ interface definition

The Test Code

The code that we would like to write to test the usage of the interface

The Generated Code

The ugly generated code that allows our tests to run

How it Works

•Recording expectations

•Verifying method calls

•Verifying arguments and returning stored values at run time

•Final verification at the end of the test

•Additional challenges…

Pitfalls

- Some custom coding needed to fit it into a TDD C++ project.
- The generator can be slow - it works best when combined with a good dependency based build tool.
- 2.0 is new and not battle tested (original version used successfully on two large-scale projects)

Where do we go from here?

- We'd like to change from "throwing exceptions when a failure occurs" to calling a "failure method" that would be supplied by a plug-in.
- This would allow the library to more easily be integrated into existing test platforms without modifying the actual generator.
- Go further with the C++ code generation idea - eliminate the need to develop directly in C++.

Thanks for Attending!

Contact Us:

- Miller@AtomicObject.com
- Pattison@AtomicObject.com

- All the code you've seen today can be found here:

www.atomicobject.com/pages/Moxy+Code+Generator