



IFS

INSTITUTE FOR SOFTWARE

Contact Info: Prof. Peter Sommerlad
+41 55 222 4984
email: ifs@hsr.ch

Mockator - Eclipse Plug-in for C++ Seams and Mock Objects



Refactoring Towards Seams

Breaking dependencies is an important task in refactoring legacy code and putting this code under tests. Feathers' seams help us in this task because they enable us to inject dependencies from outside. But it is hard and cumbersome to apply them without automated refactorings and tool chain configuration assistance. We provide support for seams with Mockator and create the boilerplate code and the necessary infrastructure for the following four seam types:

- *Object seam:* Based on inheritance to inject a subclass with an alternative implementation. Mockator helps in extracting an interface and in creating the missing test double including all used member functions.
- *Compile seam:* Inject dependencies at compile-time through template parameters. Extract a template parameter and Mockator creates the missing test double including all used member functions.
- *Preprocessor seam:* With the help of the preprocessor, Mockator redefines function names to use an alternative implementation.
- *Link seam:* Mockator supports three kinds of link seams:
 - Shadowing functions through linking order (override functions in libraries with new definitions in object files)
 - Wrapping functions with GNU's linker option -wrap (GNU Linux only)
 - Run-time function interception with the preload functionality of the dynamic linker for shared libraries (GNU Linux and MacOS X only)

Creating Test Doubles

Mockator offers a header-only mock object library and an Eclipse plug-in to create test doubles in a simple yet powerful way. It leverages new C++11 language facilities while still being compatible with C++03. Features include:

- Mock classes and free functions with sophisticated IDE support
- Easy conversion from fake to mock objects that collect call traces
- Convenient specification of expected calls with C++11 initializer lists or with Boost assign; support for regular expressions to match calls

IFS Institute for Software

IFS is an Institute of HSR Rapperswil, member of FHO University of Applied Sciences Eastern Switzerland.

<http://ifs.hsr.ch/>

Our products:

- *CUTE* - "Green Bar for C++"

<http://cute-test.com>

- *Includator* - "Static Include Analysis for Eclipse CDT"

<http://includator.com>

- *Linticator* - "Flexe/PC-Lint Integration for Eclipse CDT"

<http://linticator.com>

- *SConsolidator* - "SCons Build Support for Eclipse"

<http://sconsolidator.com>

Eclipse update site for installing an alpha version of Mockator:

<http://www.mockator.com/update>

```
struct Die {
    int roll() const {
        return rand() % 6 + 1;
    }
};

template<typename Dice = Die>
struct GameFourWinsT {
    void play(std::ostream& os = std::cout) {
        if (die.roll() == 4) {
            os << "You won!" << std::endl;
        } else {
            os << "You lost!" << std::endl;
        }
    }
private:
    Dice die;
};

typedef GameFourWinsT<> GameFourWins;

void testGameFourWins() {
    INIT MOCKATOR();
    static std::vector<calls> allCalls { 1 };
    struct MockDie {
        const size_t mock_id;
        MockDie(): mock_id { reserveNextCallId(allCalls) } {
            allCalls[mock_id].push_back(call { "MockDie()" });
        }
        int roll() const {
            allCalls[mock_id].push_back(call { "roll() const" });
            return 4;
        }
    };
    GameFourWinsT<MockDie> game;
    std::ostringstream oss;
    game.play(oss);
    ASSERT_EQUAL("You won!\n", oss.str());
    calls expectedMockDie = { { "MockDie()" }, { "roll() const" } };
    ASSERT_EQUAL(expectedMockDie, allCalls[1]);
}
```