Making Live Programming Practical by Bridging the Gap between Trial-and-Error Development and Unit Testing

Tomoki Imai
Tokyo Institute of Technology, Japan
imai.t.af@m.titech.ac.jp

Hidehiko Masuhara
Tokyo Institute of Technology, Japan
masuhara@acm.org

Tomoyuki Aotani
Tokyo Institute of Technology, Japan
aotani@is.titech.ac.jp

Abstract

Live programming environments are powerful experimental tools that enable programmers to write programs in a trial-and-error way thanks to its quick feedback. Since the feedback includes intermediate data such as a control flow and a history of variable bindings, the live programming environments integrate debugging into editing. One of the disadvantages of such interactive systems is that tests are transient. If we wrote persistent tests using an automated testing framework like JUnit, we could not fully enjoy "liveness." This is because we need to write proper parameters and expected values in advance.

We develop Shiranui, a live programming environment with unit testing features. In Shiranui, the programmers can check functions' behaviors in a lively manner and then convert the results into persistent test cases. One of the features enables the programmers to make a test case from an intermediate result that are found in a debugging process. It makes constructing error-reproducing-tests easier.

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments—Interactive Environments

General Terms Languages, Human Factors

Keywords Live Programming, Testing, Debugging

1. Proposals

We propose a set of features that enable unit testing in a live programming environment without losing its liveness. As a proof of concept, we develop live programming environment Shiranui and implement our proposals on top of it.

SPLASH Companion'15, October 25–30, 2015, Pittsburgh, PA, USA
ACM. 978-1-4503-3722-9/15/10...$15.00
http://dx.doi.org/10.1145/2814189.2814193

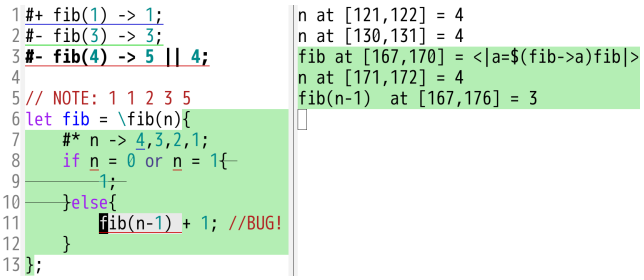

Figure 1. Screenshot of Shiranui

From the viewpoint of live programming research, our proposals enable live programming for practical software development, which is typically difficult for other live programming environments.

From the viewpoint of unit testing research, we propose novel techniques for making test cases in an interactive way, and for making test cases for anonymous functions by using intermediate execution results.

1.1 Overview of Shiranui

Shiranui is a live programming environment similar to YinYang [2] and Apple Swift [3]. It watches changes in the source code editor, re-executes the whole program immediately when it detects a change, and shows the final and intermediate results of the execution.

Figure 1 is a screenshot of Shiranui, consisting of a source code editor (left) and an environment view (right). The first three lines in the editor are called *flylines*, which serve as experimental expressions or test cases. An experimental expression is what the programmer wants to check its behavior. A test case is a pair of an expression to evaluate and its expected value. The difference between the two is just whether the programmer sets its expected value or not. The flylines also serve as documentation of functions.

We newly design not only the user interface, but also the language. Shiranui language is a dynamically-typed functional language with a domain-specific language for serializing compounded data.

```
#+ 1 -> 1;
#- calc_in_mod_n(4) -> 2;
#- <|$(n->3)multi|>(4,5) -> 2;
#- <|$(n->11)add|>(3,4) -> 7;
#- <|$(add->$(n->4)add,multi->$(n->4)multi)complex_calc|>(4,4,3) -> 0;
let calc_in_mod_n = \calc_in_mod_n(n){
    let multi = \multi(a,b){
        (a*b) % n;
    };
    let add = \add(a,b){
        (a+b) % n;
    };
    let complex_calc = \complex_calc(a,b,c){
        add(multi(a,b),multi(b,c));
    };
    complex_calc(3,5,3);
};
```

Figure 2. Test Cases for Nested Functions

1.2 Testing Features in Shiranui

Unlike other live programming environments[1], Shiranui supports the following features to bridge the gap between trial-and-error development and unit testing:

1. an editor command to convert experimental expressions to persistent unit test cases, and
2. an editor command to employ an intermediate result of an execution as a part of a test case.

With the first feature, the programmer can develop a function with experimental expressions to examine the function's behavior, and once if it shows an expected behavior, he or she can promote it to a new test case.

With the second feature, making test cases becomes a part of process for debugging and even nested or anonymous functions can be directly tested. It lets the programmer extract a function call with run-time arguments in an execution log as a new test case. This technique is useful for debugging because it can generate small subproblems. Shiranui has a domain-specific language to represent data structures and function objects in a printable form even if the data has cycles, sharing, or references to lexical variables. It enables the programmer to create test cases involving nested or anonymous functions as shown in Figure 2.

2. Implementation

Shiranui consists of the user interface, the language interpreter and a server that connects the user interface with the interpreter. The user interface is implemented as an Emacs plugin (900 LoC). The interpreter and server are developed in C++ (7,200 LoC). Emacs sends an event to the server each time the user hits a key. The interpreter evaluates a program and the server sends back a result to Emacs.

Shiranui is an open-source software available at `https://github.com/tomoki/Shiranui`.

[1] Apple announced that Apple Swift 2 and its live programming environment called Playgrounds will be able to "Create new tests and verify they work before promoting into your test suite [3]." We independently proposed our testing features in Shiranui [1].

3. Presenter

Tomoki Imai is a master's course student at Department of Mathematical and Computing Sciences, Tokyo Institute of Technology. He is a project leader of Shiranui and implemented the entire system.

4. Contents of Demonstration

We demonstrate how Shiranui's live programming features helps the programmers in software development. In addition, we illustrate what makes Shiranui different from existing interactive systems and unit testing frameworks.

Below is a scenario of the demonstration.

4.1 What is Live Programming?

First, we introduce live programming and its advantages by using a few examples.

4.2 Why (Existing) Live Programming is NOT Usable for Practical Development?

Next, we explain the motivation and challenges, namely

- live programming for practical software development, and
- testing frameworks in existing live programming environments.

4.3 Our Solution and Design of Shiranui

We show our proposals along with a demonstration of Shiranui. From the demonstration, the audience will see a

- design of unit testing features for live programming.

4.4 Free Live Coding Demos with Shiranui

Finally, we close our demonstration with a live coding session. We expect some advice, questions from the audience. We plan the following functions as subjects:

- simple arithmetic functions (e.g., power, factorial, and Fibonacci numbers),
- higher-order functions (e.g., repeat, foldr, map, and filter), and
- algorithmic problems.

References

[1] T. Imai, H. Masuhara, and T. Aotani. Shiranui: Test-friendly Live Programming Environment. In *The 31st JSSST Annual Conference*, Sep. 2014.

[2] S. McDirmid. Usable Live Programming. *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software - Onward! '13*, pages 53–62, 2013.

[3] Apple Inc. Swift - Overview - Apple Developer. `https://developer.apple.com/swift/`. Accessed 2015-6-30.