

Rescue Legacy Code With Verdé

Now you can automate characterization tests and jump start your Test Driven Development (TDD). Verdé allows you to efficiently *Rescue Legacy Code* by automating the process of creating characterization tests. These tests help you jump start your TDD and Continuous Integration.

What are we talking about?

Legacy Code Rescue – What is Legacy Code? Our definition is any code without a test (in other words, any code we are afraid to change). This can even be Java code someone wrote this morning.

Characterization Testing – A testing approach that starts by creating tests to characterize the behavior of the existing application code. This removes the fear of change, empowering developers to refactor, fix defects or add new behaviors.



Improve Time to Market

Reduced Effort to Characterize Existing Code

– Many agile teams manually characterize code today to enable refactoring. Verdé accelerates this process.

Increased Reuse

– When developers fear change, reuse does not occur. Verdé empowers developers to refactor and then reuse existing code to dramatically increase their productivity.

Reduce Cost

Reduced Rework Cycles

– Time spent fixing defects and retesting is wasteful and preventable. Verdé helps prevent regression errors by generating tests that protect the current behavior.

Reduced Maintenance Effort

– Code without tests is hard to analyze, modify, and enhance.

Reduce Training Expense

– Tests enable new developers to support old code. This validation harness is much more valuable, in the long run, than any conversations or documentation.

Improve Quality

Quality Code

– Getting a framework of trust around your existing code will empower your development team to own quality and reduce the regression issues sent to QA.

Clean Code

– To achieve clean and easy to maintain solutions, you need to be free to refactor your code. Verdé enables this refactoring.

Agile Enablement

– TDD relies on test cases to enable developers to make future changes. Verdé adds these test cases to legacy code.



Verdé Features

Verdé provides a rich feature set out of the box and is designed to be extendable and pluggable to support characterization testing needs.

Dynamic Recording of Method Calls

Non-intrusive Recorder Injection

– Verdé injects recorders into Java applications without modifying the source code.

Flexible Recording Targeting

– Verdé can be targeted at specific methods, classes, packages or across your application.

Support for Complex Class Instantiation and Setup

– Enterprise applications rely on multiple mechanisms for instantiating, injecting, and setting up objects. Verdé will allow you to use custom setup strategies. In short, the tool can adjust to enterprise architectures.

Characterization Test Generation

JUnit Generation Support

– Verdé supports multiple JUnit versions (including JUnit 3 and JUnit 4) and we continue to add adapter support for other testing tools.

Easily Maintained Using Current Skills and Tools

– Generated Tests contain easy to read code that looks much like the code you use in your hand-written JUnit tests. This means all the maintenance techniques you use today and need to use moving forward into Test Driven Development are seamlessly supported.

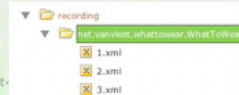
Custom Assert Code Support

– Verdé will allow you to create custom assertion strategies to validate fields not easily accessible and validate the external effects to the method being tested (i.e. file system or database changes).

Configure

```
<bind pointcut="
call(* *.WhatToWearService->*(..))
AND
!within(*.WhatToWearService)
">
<advice name="aroundCallAdvice" aspect="
..."/>
```

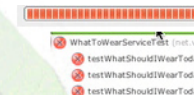
Record



Generate

```
public void testWhatShouldIWearToday_1 ()
...
// Set up mocks for the method call
String param = "Hilland";
// Call method under test
String actualReturn = target.whatSho...
```

Validate



Isolate

```
CityDAO cityDAOMock = createMock(CityDAO.class);
String mockParam0 = "Hilland";
CityData temp1 = (CityData) new CityData();
temp1.setLatitude(46.937);
temp1.setLongitude(-83.167);
temp1.setLongCityName("Hilland");
cityDAOMock.setLatitude(temp1);
expect(cityDAOMock.getData(mockParam0)).andReturn(
weatherDAOMock = createMock(WeatherDAO...
```

Mock Generation Support

Generate Tests that Contain Mocks

– Creating isolated tests is a key aspect of proper testing practices. Configuring and generating mock objects with Verdé employs the same easy to use process that is used with test case generation.

Customizable Injection Strategies

– The ways to inject a mock object vary from application to application. Verdé is designed to adjust to your application architecture.

Generate Mocks for Existing Tests

– Often, existing tests do not have good isolation and therefore take too long to run or are not easily repeatable. Verdé can be used to retrofit these test cases with mocks.

Customizing Verdé

Create Custom Test Case or Mock Generator

- Support for generation of stubs or other mocking strategies (i.e. JMock IT)
- Pluggable support for other testing frameworks (i.e. TestNG, Fitness)



To learn more about Verdé, call

1-888-374-5527

www.pillartechnology.com