# How do I test-drive GWT development?

Asked 15 years, 9 months ago Modified 7 years, 11 months ago

Viewed 2k times

5

Just googling 'TDD' and 'GWT' easily lead to this article where the author explained how he can test a GWT application without a container. However, I think his example is not test-driven as he has all the design first and then write the test afterwards, not 'test-first'.

This leads me to think: Is it possible to have 'test-first' development on a UI like GWT? Some people said UI code is not suitable for TDD. But I think by adopting the MVC pattern, maybe we can at least test-drive the MC part? (so V is the UI part which cannot be developed test-driven).

What will be the first failing test we would write on the article example?

gwt user-interface tdd

Share

Improve this question

Follow

edited Jan 13, 2017 at 18:57

Peter Mortensen

31.6k ● 22 ● 109 ● 133

asked Mar 23, 2009 at 15:27

jackysee
2,061 ● 4 ● 32 ● 39

## 2 Answers

Sorted by: Highest score (default)

14

Test driving UI is problematic because you often don't know what you want on the screen until you see it on the screen. For that reason, GUI development tends to be massively iterative and therefore very difficult to drive with tests.

This does not mean that we just abandon TDD for GUIs. Rather, we push as much code as we possibly can out of the GUI, leaving behind only simple wiring code. That wiring allows us to make the massively iterative changes we need, without affecting the essence of the problem.

This technique was probably best described by Michael Feathers some years ago in an article entitled "The Humble Dialog Box". It is also the fundamental idea behind the Model-View-Presenter pattern that caused such a stir four years ago; and has now been split into the Passive View and Supervising Controller patterns. The article link in this question takes advantage of these ideas, but in a test-after rather than a test-driven way.

The idea is to test drive everything except the view. Indeed, we don't even need to write the view for a good long time. Indeed, the View is so absurdly simple that it probably doesn't need any kind of unit tests at all. Or if it does, they can in fact be written last.

To test drive the Supervising Controller you simply make sure you understand how the data will be presented on the screen. You don't need to know where the data is, or what the font is, or what color it is, or any of the other cosmetic issues that cause the massive iteration of GUIs. Rather, you know one data item will be some kind of text field. Another will be a menu, still another will be a button or a check box. And then you make sure that the View can ask all the questions it needs to ask to get these items rendered correctly.

For example the text box may have a default value. The View should be able to ask for it. The menu may have some items greyed-out. The View should be able to ask for this information. The questions that the view asks are all about presentation, and are devoid of business rules.

By the same token, the view will tell the Supervising Controller when anything changes. The controller will modify the data appropriately, including any kind of validation and error recovery, and then the View can ask how that data should be presented.

All of this can be test driven because it's all decoupled from the visual display. It's all about *how* the data is manipulated and presented, and *not* about what it looks like. So it doesn't need to be massively iterated.

Follow

answered Mar 23, 2009 at 18:56

Uncle Bob

**4,750** ● 30 ● 19

---

3

I've successfully test-driven the development of Swing and GWT applications through the GUI.

Testing "just behind the GUI" ignores the integration between the model code and the GUI components. The application needs to hook up event handlers to display data in the GUI when the model changes and receive input from the GUI and update the model. Testing that all those event handlers have been hooked up correctly is very tedious if done manually.

The big problem to overcome when testing through the GUI is how to cope with changes to the GUI during development.

GWT has hooks to help with this. You need to set debug IDs on the GWT widgets and import the DebugID module into your application. Your tests can then interact with the application by controlling a web browser, finding elements by their id and clicking on them or entering text into them. Web Driver is a very good API for doing this.

That's only the start, however. You also need to decouple your tests from the structure of the GUI: how the user navigates through the UI to get work done. This is the case whether you test through the GUI or behind the GUI against the controller. If you test against the controller, the controller dictates the way that the user navigates through the application's different views, and so your test is coupled to that navigation structure because it is coupled to the controller.

To address this, our tests control the application through a hierarchy of "drivers". The tests interact with drivers that let it perform user-focused activities, such as logging in, entering an order, and making a payment. The driver captures the knowledge of how those tasks are performed by navigating around and entering data into the GUI. It does this by using lower-level drivers that capture how navigation and data entry is performed by "gestures", such as clicking on a button or entering text into an input field. You end up with a hierarchy like:

- User Goals: the tests verify that the user can achieve their goals with the system and demonstrates how those goals are achieved by a sequence of...
- User Activities: things the user does through the GUI, represented as drivers that perform...
- Gestures: low level mouse and keyboard input to control the GUI.

This hierarchy that is often used in the user-centered design literature (although with different terminology).

Share Improve this answer Follow

edited Jan 13, 2017 at 17:43

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered May 12, 2009 at 15:52

Nat

**9,951** ● 3 ● 32 ● 33