A Flexible Tool Suite for Change-Aware Test-Driven Development of Web Applications

Esteban Robles Luna
LIFIA. F. Informática, UNLP
La Plata, Argentina
Also at CONICET

Juan Burella
DC. F. Cs Exactas, UBA
Buenos Aires, Argentina
Also at CONICET

Julián Grigera
LIFIA. F. Informática, UNLP
La Plata, Argentina

Gustavo Rossi
LIFIA. F. Informática, UNLP
La Plata, Argentina
Also at CONICET

erobles@lifia.info.unlp.edu.ar, jburella@dc.uba.ar, {juliang, gustavo}@lifia.info.unlp.edu.ar

ABSTRACT
Though Web Applications development fits well with Test-Driven Development, there are some problems that hinder its success. In this demo we present a tool suite to improve TDD; the suite supports the representation of web requirements using a domain-specific language and the automatic generation of interaction tests among others.

Keywords
Web engineering, TDD, Web requirements, Change management.

1. INTRODUCTION
Test-Driven Development (TDD) [2] is well suited for Web applications because of its features: it is agile, it uses tests as requirements artifacts, and also uses them to determine what requirements have been fulfilled. However, traditional unit testing fails to provide quick feedback to stakeholders about interaction and navigational requirements (i.e. those that affect look and feel and represent the very nature of most Web applications). Additionally, as navigation and interaction requirements change rapidly and often, there is a need to improve change management to automatically update the test suite and simplify application evolution**.** By capturing requirement changes and deriving traceability links between requirements and the software components that fulfill those requirements, we can use change objects to upgrade the application under development.

In [4] we presented WebTDD, an improvement of TDD aimed at Web software. Our approach follows the basic TDD principles, but instead of driving the development from handcrafted unit tests, we start the process from automatically generated interaction tests, which capture the way users interact with the application and also help to outline the navigation and business models.

Due to the gap between requirements (e.g. expressed in use stories [3]) and tests, some customer requirements might remain unchecked. To bridge this gap, and considering the nature of Web applications, we have devised a domain-specific language (DSL) called WebSpec. WebSpec is used to capture interaction and navigation requirements. Its diagrams have a two-fold objective: they formalize navigation and interaction requirements, and they serve to automatically generate a suite of interaction tests that the final application must pass. We complement these diagrams with Mockups (stub HTML pages).

As shown in Fig. 1, WebTDD follows a sprint based process; in each sprint a set of requirements is implemented. We first capture requirements (Sect. 2) and use them to simulate the application. Then, we automatically generate a set of interaction tests (Sect. 3) that the application must pass. When we capture requirements we record the changes (Sect. 4) as first class objects and use them to improve the implementation phase. In this paper we present our tool suite to support WebTDD. Specifically we show:

- How to express navigation and interaction requirements, simplifying the discussion with stakeholders.
- How tests are derived automatically from requirements.
- How changes in requirements are captured and then used to improve the development cycle.

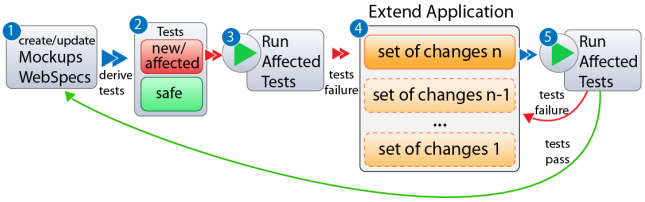

Figure 1. WebTDD approach

2. CAPTURING REQUIREMENTS
The development cycle starts by capturing requirements with Mockups and WebSpec diagrams (Step 1 of Fig. 1). Mockups help to agree on the application look and feel and WebSpec allows to specify navigation, interaction and user interface aspects in a more formal and comprehensive way (than, for example, user stories). A WebSpec diagram can be derived either from use cases or usage scenarios or stories. Similarly, mockups can be created using modern tools like Balsamiq [1]. WebSpec is independent of these technologies as long as the user interface elements can be referenced using an ID based location (e.g. button with id = "search"). WebSpec has two key elements: *interactions* and *navigations*. An *interaction* represents a point where the user can interact with the application by using the *interaction*'s widgets. A diagram has a starting interaction represented with dashed lines. Some actions (like clicking a button) might produce *navigation* from one *interaction* to another. These actions are written in an intuitive DSL with the syntax: *var := expr | actionName(arg1,... argn)*. We associate a mockup to each *interaction* to allow switching between the formal description and the proposed user interface while discussing with the stakeholders.

ICSE '10, May 2-8, 2010, Cape Town, South Africa

An example of a WebSpec diagram as produced by the tool suite is shown in Fig. 2. To express the properties the application must hold, we add invariants (Boolean predicates) to the *interactions*. For instance, the Home *interaction* (not shown for clarity) must satisfy: *Home.tweets = ${tweets}* which states that the value shown in the tweets label should be equal to the number of tweets variable (see the navigations where the variable is updated).

Using the mockups together with the actions and properties the application must hold, we derive a set of simulations that begin from the starting interaction. Each simulation opens a browser and reproduces a specific path executing actions and showing labels of the expected behavior of the application. Stakeholders can use them to propose changes before the implementation stage.

The WebSpec Eclipse plugin provides an environment to create WebSpec diagrams and to simulate them in a real browser.

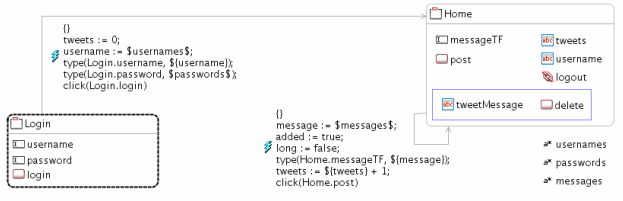

Figure 2. Tweet WebSpec diagram

3. REQUIREMENTS VALIDATION

In the 2nd step of Fig. 1 we automatically generate a set of interaction tests from the WebSpec diagram. An interaction test is a test that pops a Browser and executes a set of actions on it, in the same way a user would do. This kind of tests allows making assertions on UI elements based on their location, so we can check the values of the different widgets. We can also automatically verify whether a requirement has been successfully implemented by validating that the application passes all tests.

For each WebSpec diagram, we derive a test suite. Each path depicted in the diagram will be translated into a test case that will be named as the complete path's trail. If the diagram is cycled, we obtain finite paths by pruning to a specific length. A test case will follow the actions specified in the path, and assertions will be generated from the invariants of every *interaction*. The sentences (assignments or actions) on *navigations* will be translated to sentences in the test, such as typing text into a text field or clicking buttons. Reaching an *interaction* will require that we check its invariant (if any), by generating assertions on the test. As different *interactions* may alter the variables bound to an invariant, it is necessary to repeat the updated assertions after navigating to the same *interaction* more than once.

The WebSpec Eclipse plugin supports tests generation to Selenium [5] but other testing frameworks could be easily added by extending the generation algorithm.

4. EVOLUTION

Evolution of applications starts with changes in the requirements, and navigation/interaction requirements changes are specially frequent during the development process. WebSpec can suffer different changes, such as the addition or deletion of an *interaction* or *navigation*. An *interaction* can be modified too by the addition or deletion of widgets, changes in invariants, etc. Regarding *navigations*, we can change its preconditions or the actions that triggers them. All types of changes have been reified as first-class change objects that could be used to improve the tool's traceability features and automate some of these changes in the implementation. The WebSpec editor captures the changes made to the diagrams and stores them in files to be latter use.

To improve the development cycle (Step 4 of Fig. 1) the suite includes a change management tool that allows the manipulation of these change objects to automate the effects of changes on concrete application's artifacts. The mechanics of these effects depend on the underlying implementation setting (GWT, WebRatio [6], etc) thus we have handlers for each particular case.

As an example, let us suppose that an interaction has been added to a WebSpec, so we create the corresponding Web page for this *interaction.* When a change modifies an interaction structurally, the page that represents this interaction must be modified accordingly; Figure 3 shows how a text field element is added as the effect of the addition of a text field in the interaction. If any widget attribute is changed, the effect on the page can be automatically updated too. The tool suite has effect handlers for GWT, Seaside and WebRatio, but new ones can be easily implemented.

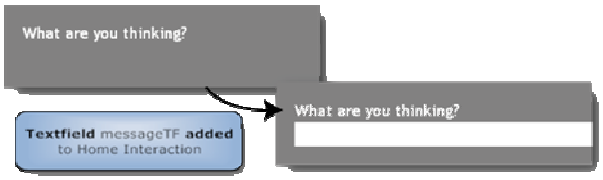

Figure 3. Text field added to the Home interaction

5. CONCLUSIONS

We have shown an agile approach for Web applications development and briefly described its supporting tool suite. Our WebTDD tool suite allows us to visually specify navigation and interaction requirements, automatically simulating the application according to requirements and generating navigation tests to validate these requirements. Changes are reified using "first class" objects, and using a change management tool we can manipulate these change objects, making them very useful in the development process. We have shown how to improve the development process by automating the effect of presentation and navigation changes.

6. REFERENCES

[1] Balsamiq. http://www.balsamiq.com/products/mockups

[2] Beck, K. 2002 Test Driven Development: by Example. Addison-Wesley Longman Publishing Co., Inc.

[3] Jeffries, R. E., Anderson, A., and Hendrickson, C. 2000 Extreme Programming Installed. Addison-Wesley Longman Publishing Co., Inc.

[4] Robles Luna, E., Grigera, J., and Rossi, G. 2009. Bridging Test and Model-Driven Approaches in Web Engineering. In Proceedings of the 9th international Conference on Web Engineering. Lecture Notes In Computer Science, vol. 5648. Springer-Verlag, Berlin, Heidelberg, 136-150.

[5] Selenium web testing system. http://seleniumhq.org/

[6] The WebRatio Tool Suite. http://www.Webratio.com.