Web Systems Acceptance Tests and Code Generation

Eduardo Aranha[1] and Paulo Borba[2]
Informatics Center
Federal University of Pernambuco
Recife, Brazil

Introduction

In Extreme Programming (XP) [2], acceptance tests are used to prove that the application works as the customer wishes. The available test languages offer low level of abstraction and legibility, because they are based in languages like Visual Basic and XML. GUI capture and playback tools facilitate the creation of test cases, though they have many limitations to program and maintain the test cases [1].

Acceptance tests interact with the GUI (Graphical User Interface) of the system, simulating the actions of users and verifying the information content presented. In Web systems, for example, the GUI is composed of Web pages and its components, like frames, links and images. In that way, the information about the GUI structure and behavior of a system can be found and extracted from its acceptance test cases, making possible the generation of part of the GUI code.

This paper presents a language and an environment to program Web Systems acceptance test cases. Code generators are presented to improve productivity and to motivate the XP practice of creation of these tests before the implementation of the proper system.

The WSat Language

The language we defined, WSat (Web System Acceptance Test), aims at a high level of abstraction and reuse, explicitly expressing aspects related to the GUI structure of the tested systems like, for example, Web pages, forms, links and texts. This is done by defining types that represent web components. In the Figure 1, we can see the initial and response page of a simple search system of Web documents.

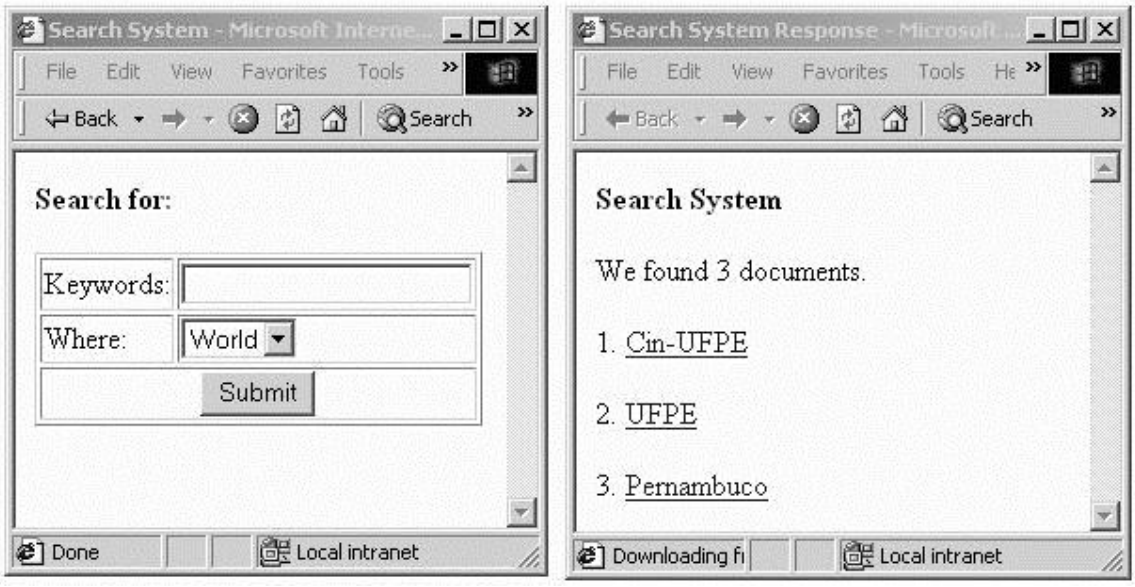

Fig. 1 – Initial and response pages of a search system.

[1] Supported in part by IPAD. Electronic mail: ehsa@cin.ufpe.br.
[2] Supported in part by CNPq, grant 521994/96-9. Electronic mail: phmb@cin.ufpe.br.

To test this system, we initially define the type `InitialPage` to represents Web pages with title `“Search System”` and an HTML form as defined by the type `SearchForm`:

```
static WebPage InitialPage {
    title = “Search System”;
    SearchForm searchForm;
    ...
}
WebForm SearchForm {
    name = "searchForm";
    method = "POST";
    EditBox {
        name = "keywords";
        value = "";
    } keywords;
    ...
}
```

WSat have predefined types like `WebPage`, `WebLink` and `WebForm`. The `WebPage` type, for example, represents all possible Web pages. The defined type `InitialPage` represents all possible Web pages that satisfy its defined properties. To test the response page of the system, we define the type `ResponsePage`, as shown bellow.

```
WebPage ResponsePage {
    title = “Search System Response”;
}
```

As we can see, we do not use the WSat keyword `static` in the definition of the type `ResponsePage`. This keyword indicates Web pages that are not generated dynamically by technologies like Servlets or JSP. This and others information not shown here are used only for code generation purpose. To verify the dynamic content of Web page and the system behavior, we create test cases as shown bellow.

```
testCase testSearchSystem {
    String url = “http://www.searchsystem.com”;
    InitialPage page = [InitialPage] getWebPage(url);
    SearchForm form = page.searchForm;
    form.keywords.value = “ufpe”;
    ResponsePage resp = [ResponsePage] form.submit();
    WebLink link = resp.findWebLinkByURL(“http://www.ufpe.br”);
}
```

The test case `testSearchSystem` requests the page at URL `"http://www.searchsystem.com"`, verifying if it conforms to the initial system page (`[InitialPage]` operator). Then, the test simulates the form submission with the `“ufpe”` keyword by calling the `submit` service defined in the `WebForm` type. To verify if the system give the correct answer, we look for the link `“http://www.ufpe.br”` in the response page.

As we can see, properties defined in WSat types are used to test the components of Web systems. In order to simulate the users actions, we can use the services of the WSat predefined types. Some of these services are used to test dynamic content of Web pages. We can use, for example, services like `findWebImageByName`, `findTextByRegExp` and `findWebLinkByURL` to retrieve components that represent images, texts and links with the given properties.

In order to validate WSat, we created an execution environment for it by compiling WSat programs to Java code.

Code Generators

In order to reduce development efforts with tests, we implemented two code generators. The first one is a test code generator, which generates WSat code from HTML prototypes used to validate the requirements. WSat types are generated to represent the components found like Web pages, frames, forms and links. As we can see, a lot of code to test GUI structure is generated. However, the code to test the system behavior could not be generated yet by this test code generator.

WSat types contain information about the GUI structure of the tested system. From this information, we can generate part of the GUI code using a system code generator. For example, considering GUIs implemented with Servlets, we can generate one Servlet for each Web page tested by a WSat type in the test code. The generated Servlets could be associated to response templates based in the HTML prototypes. Unit test classes for the Servlets and other types of code are generated, too.

The system code to be generated is dependent of the development environment used. For this reason, the system code generator was build following the Visitor design pattern [3]. Each visitor manipulates the syntactic tree of WSat programs and it has a specific functionality, like to generate Servlets or to generate JSP files. In this way, we can specialize the code generator to new development environment building new visitors.

Development Methodology

Aiming an efficient use of WSat and the code generators, some activities need to be added to XP methodology. In Figure 2, we can see the proposed changes in the XP flow.

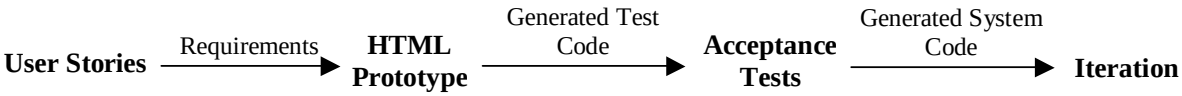

Fig. 2 – Changes in the XP flow chart.

HTML prototypes are created from the requirements found in the user stories. Then, the test code generator is used to generate part of the acceptance test code. The test programmer completes the WSat code needed in the actual iteration. The generated WSat types are complemented and new types could be created to test more complex information. The test cases are written at this time, too.

From the WSat types created in the actual iteration, we generate the part of the GUI code to be developed using the system code generator. The generated code is afterwards used by the programmer to start the system development for that iteration. With few adjustments, the code generated for the system could be executed just like the HTML prototype. The programmer is responsible now for implement the system functionalities basically writing the code under the GUI layer.

Conclusions

Through experiments, we evidence that the type definitions written in WSat code has a high level of abstraction and readability, facilitating the test programming. The use of types to represent Web components becomes the test activity more interesting and partially similar to modeling activities, eliminating part of the traditional tedium existent in writing test cases.

Programs written in WSat could check the components and the behavior of Web systems GUI, given the supporting needed to do acceptance tests. To support other types of tests, like performance and stress tests, programs WSat could have embedded Java code. However, this type of code compromises the abstraction level of code.

With the developed code generators, it is possible to generate automatically part of the test and system code, improving development productivity and motivating the creation of acceptance tests before the Web system implementation.

In one experiment done, more than 30% of test code was constituted by the declaration of WSat types (GUI structure description). The test code generator could generate a good part of that code, reducing the initial effort to program the tests. And with relation to the system code, more than 4% of it was automatically generated. The time saved by the generators, in this case, was sufficient to program simple test cases. However, it is probably not possible when we have a substantial number of test cases.

We can explore in future works the association between Web pages, like links and form actions. These associations could permit the generation of other types of code, different from the actually generated. For example, may be could be possible to generate part of the acceptance test cases.

Referências

[1] M. Finsterwalder. *Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment*. In XP2001, Sardinia, Italy.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[3] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.