GUI Development in TDD Model — Case Study

Sowmya Dhandapani*
Broadcom Limited, Bangalore, India.

* Corresponding author. Email: sowmya.dhandapani.1979@ieee.org
Manuscript submitted May 17, 2016; accepted July 14, 2016.
doi: 10.17706/jsw.11.11.1139-1144

Abstract: Test Driven Development model is identified for the quality of the final product and the ease of
development. This model has a few obstacles while applying for Graphical User Interface development, as
User Interface Testing possesses a few challenges. This paper is a case study of developing a User Interface
feature, based on the test driven development model.

Key words: UI development, test driven development, product development

1. Introduction

In product development there are different lifecycle models that can be chosen based on the need [1]. The
most commonly used model in the industry is Agile model, which allows development of the system in
functional parts, at the same time engaging the customer with feedback from the early stages. The
customization of Agile process has introduced newer and more successful models like Test driven
development (TDD)[2], Model driven development (MDD).

This paper is a case study, where the TDD was used for a UI feature development, with the help of
prototype and requirement elaboration. The intent was to verify the improvements in accepted metrics of
the software quality to measure the impact of Test Driven Development process.

2. Test Driven Development

Test Driven Development has proven advantage in software development model, as it provides a better
quality product and also allows the developer to explore the requirements well in advance to provide a
completely functional product. This development model has some limitations while applying to GUI
development. Lack of consistency is a major problem in UI Testing, where the results fail erroneously for
issues like window size, and positioning[3] and this becomes a problem while regression testing and
maintenance[4]. In this background, using Test Driven Development approach for the UI Development
becomes more challenging, where tests have to be developed even before the actual GUI is implemented.

The success of the software model depends on multiple factors. In this paper, the objective is to study
about the impact of Test Driven Development (TDD) approach on quality of the software developed. This is
measured by monitoring specific metrics, identified before and after the adoption of the new method.

3. Research Goal

Checking at the existing research, the case study [5] defines a model, where a capture and replay tool was
used to develop a set of acceptance test cases with the help of low fidelity prototypes. These acceptance
tests were then used to drive the development of the UI. The next stage planned for to go for a mixed
fidelity prototype for TDD.

Our approach was to try and use the high fidelity prototype for preparing the acceptance tests for the UI as against low fidelity sketches. This help in giving the more real time experience to the user and was possible as the product was already available – the requirement under study was to add an extra feature. The framework to introduce the UI was also already present. Hence we had to only update the new UI proposal and take it to the user.

The existing software was measured on two main criteria – Test Success Rate and Test Coverage. The test success rate is basically ensuring that the acceptance tests derived from the requirements are met. The tests were generally based on the functionality and aim to meet the requirements mentioned in the document. This also included the regression testing to ensure that the feature was not breaking existing functionality. The test coverage was to monitor the percentage of code covered during testing. This ensures that there is no unused code present which add to the software.

3.1. Existing Model

In the current software development process, developers followed up with the users through discussions and then coming up with a prototype on which the users play around and give their feedback. This was successful to get the clarity of requirements from the user, and ensure that the user and developer understand the requirements clearly. After the UI is finalized with the user, it is implemented by the developers. The feature is manually tested and approved.

Following the Agile process, this was achieved by having a story to prepare the UI prototype for the required feature, discuss and get the approval in the sprint of 2 weeks and subsequent sprints covered the implementation.

As a part of the testing the functional tests were written for the new implementation. The framework for the test modules and suite were already in place and the new tests were generally added for the new functionality/feature introduced every release. This involved writing the test code manually and adding the required tests to the regression suite. The developers were comfortable with these procedures.

As the test case writing was done after implementation the tests generally intend to be success path and aimed to provide a 100% success rate. The test coverage was 80%. The average number of test for a single UI component was 3 for 3 different use cases. The aim was to improve the quality of the tests by increasing the number of tests per component, thereby increasing the code coverage as well.

As said, to insert images in Word, position the cursor at the insertion point and either use Insert | Picture | From File or copy the image to the Windows clipboard and then Edit | Paste Special | Picture (with “Float over text” unchecked).

The authors of the accepted manuscripts will be given a copyright form and the form should accompany your final submission.

3.2. Proposed Model

Extending the current model, a new TDD based GUI development model was proposed. Continuing the existing Agile Scrum model, the prototype and the tests were to be developed in the first sprint. These tests will be failing initially due to lack of functionality. As the feature is implemented, the test pass percentage will increase. This way the Tests become the criteria that need to be satisfied for the completion of the feature in subsequent sprints. This is expected to also indirectly motivate both success and failure path testing, and error handling scenarios, thereby adding to the code coverage.

4. Implementation

The product under study was developed Netbeans based Desktop Application. These tests were run
periodically to ensure the regression health of the product between multiple feature developments. The product features were developed based on Agile Scrum Model [6]. The tests were based on Netbeans Jemmy [7] and Jelly Tools libraries which allowed the keyword, type and text based testing for the UI components [8][9]. The feature was split into smaller stories, which were executed across the sprints. This section discusses the current model, the proposed new model and the changes in the implementation and the output.

Once the user had agreed up on the high fidelity prototype, the acceptance tests were finalized. These test details and the requirement details were shared with the user to ensure that they align with the user expectations. As the acceptance tests for the new feature (henceforth referred as Feature A), were written based on the requirements and feedback from the end user. The high fidelity prototype helped in bringing out the inherent requirements like validation, error handling. These requirements were then mapped to test-cases and were implemented.

TDD requires the tests be written before the actual development. The existing framework from Netbeans to write UI tests was extended for writing the acceptance tests. While implementing the new tests, few properties used for the development had to be finalized, like the display names (labels, titles, tool-tips) and the UI component names (button, text-field, and table names) that were required for writing the tests. This decision was made easy as the high fidelity prototype gave a closer picture of the actual implementation. Most of the UI components were finalized and allowed the naming and labeling of these easier.

Once the test cases were ready, the implementation started. The implementation of every UI component could be immediately checked with the tests to avoid any regression.

5. Observations

This section discusses the observation from the implementation of the proposed model, their merits and demerits over the existing model. Figure 1 summarizes the details collected from the testing. The details are discussed based on different parameters considered.

5.1. Team

The prior knowledge of the development and testing framework helped in coming up with the prototypes and tests. The expertise of the developer helped in identifying the design that could accommodate the test based development and the changes [10]. It was obvious that this approach definitely needed the familiarity of the framework for the proposed model.

5.2. Requirements

This new TDD based approach ensured that there were appropriate tests present for every requirement. The outcome of the test case driven development was very obvious in the quality of the product. At the end of implementation the failures were minimal and all the requirements were covered.
Changes to the UI requirement at the later stage involved changing the UI tests and the UI. This was a setback when the new ones were in conflict to the original accepted requirements and extended the project by few days/weeks. However it brought out the point that the clear vision of the UI is required to ensure that the Tests written for UI does not undergo change after completing the implementation [11].

5.3. Test Coverage

5.3.1. Scope

In the case where the tests were written after implementation, the general approach was for the success path. In the approach to implement and write the tests, the intent was more to place the test that would pass, providing the success rate. The scope was to cover all the requirements which made it more elaborate. Even lower priority issues, like Help availability and its content were covered.

5.3.2. Test development

The tests were written by developer who was implementing the feature. This enabled the developer to consider of the functionality as a whole and understand the requirements before starting the design and implementation. The developer also felt that the regular validation with the acceptance tests provided greater confidence on the functionality during the implementation.

5.3.3. Type tests

The TDD allowed the coverage of negative scenario as a part of the requirement and thereby providing better coverage of functionality and code. Table 1 summarizes the support for different functionalities covered during testing before and after the TDD model. The smaller details like the help, and the preferences which are generally lost in priority were also included while writing the tests. This was visible though the code coverage. The test success rate was 100% but the number of tests covered per UI component became twice the original number to check the error conditions.

<table>
<thead>
<tr>
<th>TestCase Type</th>
<th>Before TDD</th>
<th>After TDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Exceptions and Errors</td>
<td>Partially Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Content Validation</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Help &amp; Preferences</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>

5.3.4. Implementation

As the project was carried out on an existing product, the presence of the framework facilitated the high fidelity prototype to come up more quickly and the user also got a full flow of the functionality more closer to the final product— from where to launch, how the UI looks and how to complete the required UI actions. Again the test framework was already present, which made the implementation of the tests quick and easy to ensure the requirements are met.

From the implementation perspective, the usage of code based testing reduced the chances of regression when new requirements were added. Every code change could be verified with the tests. This enabled the first functional output to be more complete and also the design to be more durable. Also the code changes during the actual testing cycle were easier to integrate due to the availability of tests for validation. The productivity of the developer improved while implementing the TDD model [10] [12].

From the original 80% code coverage rate for the tests, the new feature showed an increase to 90% which ensures that the test covers the feature completely for future regressions as well. Other functionalities impacted in this feature also showed steep increase from earlier 60% to new 90% in the coverage, as a part of this testing based development. Lines of Code also trimmed considerably as the code was aimed only to
meet the test.

5.3.5. Post-development

After the product was completed and given for beta testing there were few UI requirements that came up – to remove an extra dialog, changing the layout of a panel, adding an extra UI component. The presence of a testing framework enabled these changes to be done, without breaking the functionality – along with ensuring the completeness of the feature.

Any addition to the design that came up after the product was released was taken in as a request. These UI changes could be regressed with the tests previously written to ensure the whole functionality is maintained.

There were suggestions that came up later to change the UI. These were handled by first updating the tests and then including the required changes in the implementation.

6. Conclusion

The case study covered a small project with one developer and tester each. The study shows that the use of TDD helps in improving the test coverage and producing a lean implementation for required feature. The framework for development and testing were already present and minimal change was required for reuse. The sample was confined in the scope and resources, and cannot be used as a standard. However, this study confirms that it is feasible to use UITDD for successful development of the User Interface under specific factors, and can be used to the quality of the product delivered was better than the regular approach.

The next level of research is to explore the factors that can aid to the success of TDD based GUI development on a larger team and different prototyping options. This study paves way to understanding the effect of UI development with test model. Also, the impact of the size, expertise of the team in both development and test definition needs to be evaluated in future studies.

References


quality and productivity in a test-driven development context.

Sowmya Dhandapani works as a staff research engineer in Broadcom Limited, Bangalore India. She completed her masters in computer engineering from Illinois Institute of Technology, Chicago in 2013. She has been working in the software industry for past 13+ years, after graduating from one of the prestigious colleges in South India, in 2001. She has worked on applications related to telecommunications, data warehousing, and currently leading a team on the tool development for Broadcom Limited. She has played the role of developer, designer, team lead and consultant on many features in the products from Nortel and Oracle; her work which concentrated primarily on the Desktop User Interfaces, has helped her understanding the perspective of both the users and developers of the product. She is currently working on understanding the integration of the User Centered Design and the Software Development Cycle process to achieve a single standard that can develop a product with the advantages of both the streams.