A Model-Driven Approach for Specification-Driven Automated UI Testing

- blinded for review -

ABSTRACT
The most important key to success for a software project are concise and relevant requirements as recent research shows. Requirement documents fulfilling these quality criteria include besides others an at least semi-formal description and an abstract UI description. However, stakeholder expectations change constantly during a project and requirement documents and especially UI descriptions are often not updated leading to severe problems in the development and testing phase. This paper introduces an approach that automatically recognizes differences between requirement and UI descriptions and rewards up to date requirement documents. The center of the approach is a domain-specific language that connects textual requirement descriptions to the corresponding abstract UI and thereby enables advanced validation and user support. In addition to improving the overall quality of requirement documents, the language comes with a generator infrastructure to generate automated UI test cases for different UI frameworks such as Web and SWT. The focus of the approach is to ease the creation and validation of requirements documents incentivized by the automated UI test cases generated from the descriptions.

CCS Concepts
•Software and its engineering → Model-driven software engineering; Maintaining software;

Keywords
Domain-Specific Language, Behavior-Driven Development, Model-driven software development, Automated UI testing, Xtext

1. INTRODUCTION
- Well documented and up-to-date Requirements are crucial for project success - Many standards on how to describe requirements and what to include (functional, non-functional, UI-descriptions) - UI descriptions are an important part since they create a common ground for developers, testers and business analysts to discuss even before implementation has started - Because requirement documents play a central role in software development project every change affects multiple documents. If there is no traceability between the different artifacts it might happen that documents get not updated accordingly. Especially, in projects under time pressure documents will not be updated leading not only to outdated requirements but also to outdated documentation and more importantly outdated or irrelevant test cases. - The approach presented in this paper tries to encourage the requirements maintenance by rewarding it with fully generated automated test cases. - The domain specific language introduced combines wireframe UI descriptions with specification written in a ubiquitous language as defined by the behavior driven development process. - As for tools like cucumber the restriction applies: The main focus of the specification driven testing approach is to ensure a clean and up-to-date documentation of the system rewarded with fully generated test cases. Well understood, documented and up-to-date requirements are crucial to successfully accomplishing a software development project. There are many standards for software requirement such as [Quelle] and [Quelle] that all define the structure and the content of the textual requirement specifications and additional artifacts such as abstract UI description [Quelle for Standard]. Especially in projects following an agile process such as Scrum or Canban, requirements are documented by user stories encapsulating a specific function from a user's perspective. In addition, the Behavior Driven Development approach divides user stories in more detailed scenarios that come with a pre-defined format using an ubiquitous language to describe the expected behavior.

SAC'17, April 3-7, 2017, Marrakesh, Morocco
Copyright 2017 ACM 978-1-4503-4486-9/17/04...$15.00
http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx

2. RELATED WORK
BDD Approach Wireframing Generating automated test cases

3. SPECIFICATION-DRIVEN UI TESING

3.1 Specifying the Application
Application to create calendar entries. Assume that we focus on describing the main features of the applications frontend. Additional parts of the software described by the

agile testing pyramid such as integration tests and unit tests might be tested using other tools Login Screen Welcome Screen showing the agenda of the day and a plus button at the top right to calendar entries Write specification for logging in Feature file for some scenarios Bring business analysts and testers together to write scenarios based on the UI descriptions After the feature files have been written the final ingredient to convert the specification into an executable testcase needs to be added by the developer. The developer of the UI application has to come up with a mapping file that maps the logical screen elements from the wireframesketch to the elements in the real implementation. Introducing a sample mapping file Running the generator Executing test cases locally and in an CI environment

3.2 Composition of the Specification Language

3.2.1 *Specification Language Architecture*

Figure to show how everything interacts

3.2.2 *Wireframesketcher Integration*

Existing Framework used to specify wireframes within an eclipse environment. EMF Based to easily integrate with the Xtext based DSLs. Metamodel patched to add some marker interfaces Xtext Adapter added to make model elements referable from the outside.

3.2.3 *Feature DSL*

Feature DSL defines the structure of a feature file as well as re-usable language parts Based on the Control Action Parameter from Jubula. Possible sentences are always based on the current scope

3.2.4 *Mapping DSL DSL*

Maps logical elements from the screen to real world implementations Navigator concept Identify by ID Validations

3.2.5 *Generator*

Takes feature and mapping file in combination with the wireframesketch to generate test cases The generator infrastructure allows different generator for different UI Frameworks. In the default implementation there is a generator for Web applications In addition, there have been proprietary generators for SWT based RCP applications The generators can currently be configured per Eclipse workspace. Generator generates Cucumber specifications files that are able to run web-based test with spock as execution framework and selenium as web driver underneath. The logic to access the elements on each page is encapsulated in page objects (fowler) that are also generated from the screen files. Some of the functionality generated into the page objects relies on library methods that come with the framework. The generator can be integrated in a CI environment like Jenkins. In such an environment it will take the feature, mapping and screen files and turn them into Cucumber specification, before they are executed.

4. DISCUSSION

Not introducing a new tool or technique, but bringing existing approaches together

5. CONCLUSION

Especially after the first release of a functionality the user stories and UI description become less relevant, because discussion now focusses on the running application. Mitigate this by putting the feature description and the UI description in the focus of development Not only generate Testcases, but also generate implementation, especially in the UI area. Web-enable to increase non-technical user's acceptance. Focus on Service tests in the same way (replay UI description by formal Service Descriptions such as WSDL) Improve process inclusion (write scenarios in HP Quality Center and report execution back to HP QC). Language Design Currently little bit technical to ease scoping could be changed. Leading to advanced scoping and content assist mechanisms to achieve