SPL-AT Gherkin: A Gherkin Extension for Feature Oriented Testing of Software Product Lines

Tugkan Tuglular
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

Sercan Şensülün
Dept. of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
sercansensulun@iyte.edu.tr

***Abstract*— As cloud platforms turn into software product lines (SPLs), testing products composed of customer selected features becomes more and more important. In this paper, we propose a feature-oriented testing approach for platform-based SPLs through a novel extension to Gherkin called SPL-AT Gherkin and a novel automatic test method generation technique, which utilizes TestNG framework. We demonstrate the applicability of the proposed approach by a case study.**

Keywords*— *software product lines, feature-oriented testing, acceptance testing, Gherkin, automatic test generation

I. Introduction

Nowadays, cloud platforms with their extensions, such as web and mobile applications, are new wave of software product lines. The features selected by customers shape how the platform behaves, and the customer is billed according to the selected features. These cloud platforms enable companies to add and/or modify features much easier than the past. However, the problem of how selected features work in a reliable manner stays still. This problem gets harder if different mobile applications are provided to different segments of users of the customer.

To assure the quality of the delivered mobile applications with respect to selected features, one approach is to follow feature-oriented testing, where acceptance tests (ATs) are determined with the "definition of ready". Then selected features will indicate acceptance tests that need to be executed. In this paper, we propose a feature-oriented testing approach based on Gherkin but with a novel extension called SPL-AT Gherkin. The proposed approach also includes an automatic test method generation technique for concrete acceptance test cases.

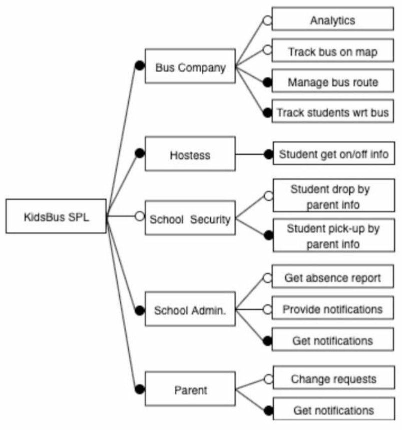


Fig. 1. SPL feature diagram for KidsBus™

Feature diagrams [1] are used to represent the feature options in software product lines for user selection. An example feature diagram is given in Fig.1 for the SPL named KidsBus™, which is chosen as case study in this work. KidsBus™ is a platform that manages the school bus transportation processes effectively and efficiently. The root of feature diagram represents the SPL and the nodes are features, which can be mandatory or optional, represented by filled circle and empty circle respectively. Product diagrams, similar to feature diagrams, are user-centric representations of product feature configurations, where all feature selections are made for the product. An example product diagram of is given in Fig.2, which shows selected features of the product called Gold KidsBus™. Since the feature selections are made, filled and empty circles are removed.

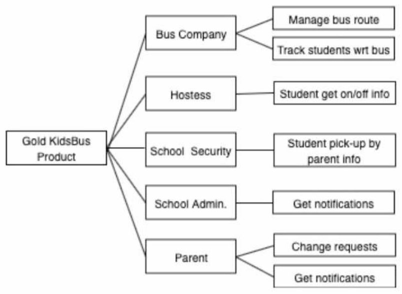


Fig. 2. SPL product diagram of Gold KidsBus™

The proposed feature-oriented testing approach based on SPL-AT Gherkin enables automatic composition of acceptance tests with respect to selected feature combination provided by the product diagram. SPL-AT Gherkin allows analysts and testers to write scenarios with tags or placeholders, which are replaced with concrete objects during test method generation. The proposed approach follows agile practices for developing software product lines proposed by de Souza and Vilain [2].

Other advantages of the proposed approach are as follows. The proposed test method generation technique is open to work with any testing frameworks. Only, implementation of the mapping rules needs to be changed with chosen framework API. Furthermore, it is possible to reuse the templates of feature-oriented acceptance tests written in SPL-AT Gherkin. In addition, acceptance test driven development can be pursued with the proposed approach.

The paper is organized as follows. After the fundamentals section, the proposed approach is explained with a running example in Section III. Section IV describes the case study and presents the results obtained along with a discussion. Related

978-1-7281-2607-4/19/$31.00 ©2019 IEEE
DOI 10.1109/COMPSAC.2019.10230

work is outlined in Section V. Section VI concludes the paper and lists possible future works.

II. Fundamentals

A. Domain Specific Languages for Test Generation

Gherkin [3] is a domain specific language to create project documentation and automated tests. It provides the behavior definitions of the intended software not only to product owners and business analysts but also to developers and testers. In other words, it is a well-known language, which is understandable by any teams with +70 spoken languages support. Gherkin is a line-oriented language in terms of structure and each line has to be divided by the Gherkin keyword except feature and scenario descriptions. In this paper, some of the Gherkin keywords, which are *Scenario Outline*, *Given*, *When*, *And*, *Then*, *Examples*, are going to be handled in describing SPL-AT Gherkin.

B. Page Object Pattern

Page object design pattern was introduced for web pages to hide user interface (UI) details from clients. It is a basic encapsulation mechanism that finds UI components such as Header or Paragraph tags in HTML pages and manipulates them without dealing with any technical details through a web driver. It is necessary to manipulate UI components when writing test against any web page. Although Martin Fowlers suggested this pattern for web pages, he claimed that it could be applied to any UI technology [4].

Right to his claim, this pattern is evolved to be used in mobile application testing domain. For instance, a login page in a mobile application contains one editable field, which is called EditText in Android or UITextField in iOS, and one button to validate written text in the editable component. When writing tests to this page, a test framework, such as Appium, can be used to manipulate these UI components. Accessor methods, such as getText() and setText(…) can be developed for editable field, and button can be manipulated by action oriented methods, such as clickButton(). Appium API methods hide technical details behind these methods. Thanks to this encapsulation, test methods can be developed with accessor and action-oriented methods without knowledge of Appium API.

C. Unit Testing Framework

Test Next Generation (TestNG) [5] is a testing framework for Java developers inspired by JUnit. It is suitable to write unit, functional, end-to-end, integration etc. tests. It works well with test automation frameworks, such as Selenium and Appium. It can be plugged to some integrated development environment such as Eclipse and Intellij IDEA.

TestNG supports important features such as data-driven testing, parametrized testing and flexible test configuration. Test methods can take one or more parameters and different arguments can be passed to same test method for different scenarios. Although parameters can be set in two different ways, through testing.xml or programmatically, testing.xml is used for the proposed technique.

Another important feature utilized in the proposed technique is priority feature of TestNG. If the order of the test case executions is critical, priority should use with @Test annotation. Priority is represented by integers and lower value is executed first.

III. Feature Oriented Testing of Software Product Line based Mobile Applications

In this paper, we propose a feature-oriented testing approach for platform-based SPLs and demonstrate its application on mobile applications of which back-end is a cloud-based application platform. In the selected case study, the features are presented through variable mobile applications instead of web pages. Customer selected features are used to develop role specific mobile applications and these mobile applications communicate with the platform where only customer selected features are enabled.

In the proposed feature-oriented testing approach, we assume that a feature may have a number of user stories and each user story may have a number of acceptance test cases, which are created manually at the requirements specification phase. A Gherkin Scenario is an abstract acceptance test case, which is instantiated once converted to unit test method(s). Gherkin is an efficient language to write User Scenarios. However, it is not sufficient to define acceptance tests with placeholders. Therefore, we extend Gherkin with a tag structure to introduce variability that SPLs assume and UI components that a mobile application utilizes. So, we called this extended Gherkin as SPL-AT Gherkin, which is explained in the following sub-section. Following that we introduce our automatic test method generation technique that inputs a SPL-AT Gherkin scenario and automatically produces TestNG classes and an XML file to run tests.

A. SPL-AT Gherkin

The proposed approach utilizes mapping rules to achieve transition from user scenarios to TestNG test project. If user interfaces, as a set of UI components, and their behavior were to be defined in user scenarios, transition would be easy. For this purpose, we propose a tag structure to achieve this transition. In this tag structure, there are two different tags, which are address sign (@) and dollar sign ($). They are added onto Gherkin to write scenarios convertible to executable acceptance tests. While @ tag is used to define UI components such as Edit Text, Button, Text View etc., $ tag is used to define their behavior.

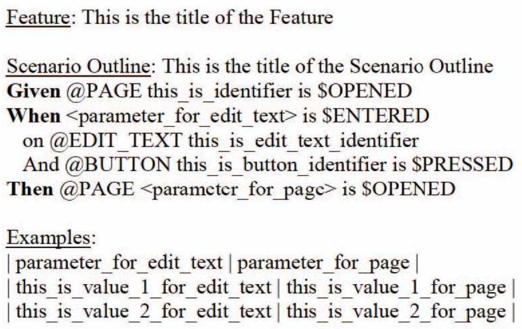
We propose a tag structure based on object-event pairs to extend Gherkin. @ tag represents objects in mobile applications, such as PAGE, BUTTON, EDITABLE TEXT (EDIT_TEXT in short), and TEXT_VIEW. For $ tag, different adjective keywords such as OPENED, ENTERED, CLICKED, ENABLED, DISABLED, and SHOWN are selected to represent events. $ tag must be used with its related @ tag. Allowed combinations are given in Table I.

TABLE I. Tag Usage Rules

PairID	Where to Use	@ Tags	$ Tags
P1	Given, When	@PAGE	$OPENED
P2	Given, When	@BUTTON	$CLIKED
P3	Given, When	@EDIT_TEXT	$ENTERED
P4	Given, When	@TEXT_VIEW	$SHOWN
P5	Then	@PAGE	$OPENED
P6	Then	@BUTTON	$ENABLED or $DISABLED
P7	Then	@EDIT_TEXT	$ENABLED or $DISABLED
P8	Then	@TEXT_VIEW	$SHOWN

@tag-$tag pairs are identified with Pair ID in Table I and Table I should be read as follows. In *Given* and *When* sections of SPL-AT Gherkin, @PAGE must be used only with $OPENED, @BUTTON with $CLIKED, @EDIT_TEXT with $ENTERED, and @TEXT_VIEW with $SHOWN. In *Then* section of SPL-AT Gherkin, @PAGE must be used only with @OPENED and @TEXT_VIEW with $SHOWN, whereas @EDIT_TEXT and @BUTTON must be used with $ENABLED or $DISABLED. Only these pairs are allowed in *Scenario Outline* of SPL-AT Gherkin to generate test methods with the correct source codes.

We consider *Scenario Outline* in SPL-AT Gherkin as acceptance test for a user story, where this user story belongs to the *Feature* in SPL-AT Gherkin. In Fig.3, an acceptance test template in SPL-AT Gherkin is given to materialize our approach. Identifiers proceeding @ tags are UI components in the mobile application page.



Feature: This is the title of the Feature

Scenario Outline: This is the title of the Scenario Outline
Given @PAGE this_is_identifier is $OPENED
When <parameter_for_edit_text> is $ENTERED
  on @EDIT_TEXT this_is_edit_text_identifier
  And @BUTTON this_is_button_identifier is $PRESSED
Then @PAGE <parameter_for_page> is $OPENED

Examples:
parameter_for_edit_text	parameter_for_page
this_is_value_1_for_edit_text	this_is_value_1_for_page
this_is_value_2_for_edit_text	this_is_value_2_for_page

Fig. 3. An example of acceptance test template in SPL-AT Gherkin

In the next sub-section, our automatic test method generation technique is explained with our mapping rules that automatically produces TestNG classes and the XML file from a SPL-AT Gherkin scenario.

B. Executable test method generation

The design we utilize in our proposed test method generation technique is based on TestNG framework and TestNG test classes are automatically generated by this design given in Fig.4. In the design of our proposed technique, there is a Base Page class and it manages Appium API methods. It has five methods, which are called as click, setText, getText, isEnabled, and isShown. All of them take an identifier parameter of type String to ensure which UI component is referred in the application under test.

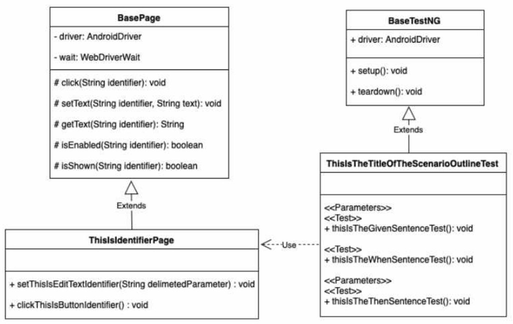

BasePage
- driver: AndroidDriver
- wait: WebDriverWait
# click(String identifier): void
# setText(String identifier, String text): void
# getText(String identifier): String
# isEnabled(String identifier): boolean
# isShown(String identifier): boolean
Extends
ThisIsIdentifierPage
+ setThisIsEditTextIdentifier(String delimetedParameter) : void
+ clickThisIsButtonIdentifier() : void
Use
BaseTestNG
+ driver: AndroidDriver
+ setup(): void
+ teardown(): void
Extends
ThisIsTheTitleOfTheScenarioOutlineTest
<<Parameters>>
<<Test>>
+ thisIsTheGivenSentenceTest(): void
<<Test>>
+ thisIsTheWhenSentenceTest(): void
<<Parameters>>
<<Test>>
+ thisIsTheThenSentenceTest(): void

Fig. 4. UML class diagram of test method method generation

The proposed test method generation technique utilizes the following mapping rules to generate executable TestNG test methods. With the help of these mapping rules, SPL-AT Gherkin scenarios are converted into Java source codes that make up the TestNG test classes.

Rule 1. Generation of Page Class

The first rule is related with child classes of the Base Page class. *Scenario Outline* has to include at least one @PAGE tag with same identifier in *Given* and *When* parts. If only one page on mobile application is tried to be tested, @PAGE tag with another identifier only could be existed into Then part of the SPL-AT Gherkin. If @PAGE tag is detected with identifier, child of the Base Page should be created into Acceptance Test Project with identifierPage name. For instance, in Fig.3, there is one @PAGE tag with this_is_identifier identifier. According to this rule, ThisIsIdentifierPage class, which is the child of the Base Page class as in Fig.4, should be created into the Project as shown in Appendix A.

Rule 2. Generation of Method for Editable Text

In the second rule, inside of child class mentioned in Rule 1 is going to be processed. @EDIT_TEXT tag can exist in *Given* and *When* parts in *Scenario Outline* with delimited parameter. When it is detected with delimited parameter, there should be a method into the child class to set any text to mentioned UI component via @EDIT_TEXT tag. Moreover, the setText(String identifier, String text) method of the Base Page should exist inside of the generated method with the given identifier from SPL-AT Gherkin. For instance, in Fig.3, @EDIT_TEXT tag exists in When part with this_is_edit_text_identifier identifier. There is also delimited parameter, that is shown with <> special characters, in When part and a value for this parameter is defined on *Examples* data table in SPL-AT Gherkin. When this rule is applied, implementation of the method is going to be generated as in Appendix A. The point is that the implementation is generated automatically so that the rule could be applied for any delimited parameter in *Scenario Outline*.

Rule 3. Generation of Method for Button

@BUTTON tag is going to be focused in Rule 3. This tag has to appear in *Given* or *When* parts of a *Scenario Outline*. A method inside the child class is generated for the clickable UI component Button. When the tag is detected with the identifier, action-oriented method has to be created inside of the child class. Then, click(String identifier), that is implemented into the Base Page super class, method should be put into this method. For instance, in Fig.3, the tag is found in *When* part with this_is_button_identifier identifier. As a result, implementation of the method similar to the one in Appendix A is generated automatically.

Rule 4. Generation of Page Classes

Definition of the Rule 4 is that every *Scenario Outline* in SPL-AT Gherkin is a sub-class of the BaseTestNG. For instance, in Fig.3, title of the *Scenario Outline*, which is "This is the title of the Scenario Outline", is mapped to the name of the class. When this rule is applied, a similar class to the one in Appendix A is generated automatically.

Rule 5. Generation of Scenario Outline Class

When any *Scenario Outline* is analyzed, three base keywords, which are *Given*, *When*, *Then*, are noticed. Moreover, each keyword describes itself with one sentence. Rule 5 indicates that each keyword is going to be converted to a TestNG test method with @Test annotation into the child of the BaseTestNG class that was described in Rule 4. So that, number of the test methods are going to be equal to number of the keywords that exist into these three base keywords. When this rule is applied the test methods in Appendix A are going to be generated.

Rule 6. Generation of Priority Assignments

There is a hierarchy between *Given*, *When* and *Then* keywords in terms of the execution order. *Given* part is described as initialization part of the scenario such as opening the application page. *When* part has some event-based operations, e.g., click button, set username in to text field. In *Then* part, some assertion operations are found, such as page is opened, or button is disabled. In summary, Rule 6 indicates that test methods, which were generated in Rule 5 based on *Given-When-Then* template, have to be executed in the same order. @priority TestNG annotation is going to be used to implement this order. When Rule 6 considered, there are three test methods in ThisIsTheTitleOfTheScenarioOutlineTest class: thisIsTheGivenSentenceTest, thisIsTheWhenSentence-Test, thisIsTheThenSentenceTest.

Rule 7. Generation of Parameterized Tests

The goal of Rule 7 is to prepare parameterized tests for the methods explained above. To achieve this, *Examples* in SPL-AT Gherkin is going to be converted to @Parameters TestNG annotation. In *Examples*, rows other than header row represent value of each cell of header row. For instance, *Scenario Outline* in Fig.3 has two different delimited parameters, namely delimited_parameter_1 and delimited_parameter_2. Values of these parameters appear in rows of *Examples*, i.e., this_is_value_for_param_1, this_is_value_for_param_2. The rule indicates that when any delimited parameter detected in *Scenario Outline*, it is to be converted to the parameter of the test method. For instance, delimited_parameter_1 is going to be defined as parameter to thisIsTheGivenSentenceTest test method as shown in Appendix A.

Rule 8. Generation of TestNG configuration file

After setting parameter annotations in the test class called ThisIsTheTitleOfTheScenarioOutlineTest by the previous rule as in Appendix A, values of these parameters should be passed to the test methods. Passing parameters values through testng.xml is one of the passing manners in TestNG framework. Testng.xml file is a configuration file to manage test suite and its parameters in any test project. With Rule 8, <test>, <parameters>, <classes>, and <class> XML tags of Testng.xml file are set. As a result of *Scenario Outline* in Fig.3, testng.xml file shown in Appendix B is generated with Rule 8. When the test class is run with the testng.xml file, test outputs will be generated.

By applying the above rule set to *Scenario Outline* written in SPL-AT Gherkin, class implementations as well as testng.xml file are automatically generated with respect to UML class diagram given in Fig. 4. For the example user story template shown in Fig. 3, the classes in Appendix A and also testng.xml file in Appendix B are automatically generated. In other words, *Scenario Outline* written in SPL-AT Gherkin is converted to an executable test project.

IV. Case Study

KidsBus™ is a platform that manages the school bus transportation processes effectively and efficiently. It provides coordination between the parents, school and bus company so that children are safely taken to their school and back to their homes. The cloud based KidsBus™ platform has the features grouped according to the roles that take place in the transportation processes. The roles are parent, school administration, school security, hostess, and bus company. Depending on the selection of the features by the school administration, a mobile application for each role is generated from the software product line. The feature diagram of KidsBus™ platform is given in Fig.1.

The proposed technique is applied to the mobile application KidsBus™ School Security and the details are explained in this section. School securities, in KidsBus™ environment, can display the list of students whom will be taken from the school by an adult. To show implementation of our proposed technique on the mobile application, SPL-AT Gherkin based acceptance tests for three different pages of School Security mobile application. These pages are getting SMS code for the entered phone number, verifying the SMS code and creating new password and they are shown in Fig. 5. Corresponding test classes and TestNG xml file are automatically generated, and finally those test classes are executed.

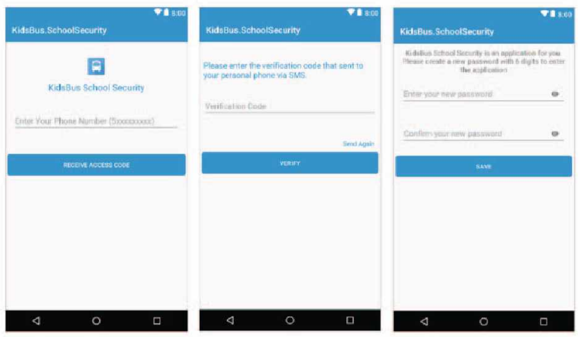

8:00
KidsBus.SchoolSecurity
KidsBus School Security
Enter Your Phone Number (5xxxxxxxxx)
RECEIVE ACCESS CODE
8:00
KidsBus.SchoolSecurity
Please enter the verification code that sent to your personal phone via SMS.
Verification Code
Send Again
VERIFY
8:00
KidsBus.SchoolSecurity
KidsBus School Security is an application for you. Please create a new password with 6 digits to enter the application
Enter your new password
Confirm your new password
SAVE

Fig. 5. KidsBus™ Mobile Application School Security Sign-up Pages

Users can enter their phone number to the editable text area and can send the number to KidsBus™ system with the button. If the phone number exists in KidsBus™ system as school security role, SMS verification code will be sent to the phone. Otherwise, the mobile application will remain on the same page with an error message. Two different test scenarios can be executed on this page. In the first scenario, phone number, which belongs to any school security role, will be entered and then it is asserted that the page is changed to the second page in Fig.5. In the second scenario, phone number, which does not belong to any school security role, will be entered and it is expected that current page will not be changed. The corresponding SPL-AT Gherkin scenario outline is shown in Fig.6.

```
Feature: Get SMS Code

Scenario Outline: Get SMS code scenario
Given @PAGE ReceiveVerificationCodeActivity is $OPENED
When <username> is $ENTERED
  on @EDIT_TEXT usernameInput
  And @BUTTON loginButton is $CLICKED
Then @PAGE <page> is $OPENED

Examples:
| username | page |
| "5454339401" |
 ".Activity.CommitVerificationCodeActivity" |
| "5359144691" |
 ".Activity.ReceiveVerificationCodeActivity" |
```

Fig. 6. Acceptance test for "Get SMS code" scenario in SPL-AT Gherkin

On the verifying the SMS code page of mobile application, there are two different user interface components, which are an edit text and a button. Users, who has school security role in KidsBus™ system, should enter the verification code, which is sent via SMS to their phone, to create user password on the third page, namely creating new password page. To test this feature, KidsBus™ system generates same verification code for all test users. So that, mobile application test project does not need to read content of the SMS. In other words, mobile application test project assumes that verification code is 112233, if the phone is verified by KidsBus™ system as school security role. Two different test scenarios shown in Fig.7 will be executed as valid and invalid verification code. These scenarios could be extended with different verification code combinations.

```
Feature: Verify SMS Code

Scenario Outline: Verify SMS code scenario
Given @PAGE ReceiveVerificationCodeActivity is $OPENED
When <username> is $ENTERED
  on @EDIT_TEXT usernameInput
  And @BUTTON loginButton is $CLICKED
Then @PAGE CommitVerificationCodeActivity is $OPENED
  And <passcode> is $ENTERED
  on @EDIT_TEXT activation_code
  And @BUTTON loginButton is $CLICKED again
  And @PAGE <second_page> is $OPENED

Examples:
| username | passcode | second_page |
| "5454339401" | "111111" |
 ".Activity.CommitVerificationCodeActivity" |
| "5454339401" | "112233" |
 ".Activity.CreateNewPasswordActivity" |
```

Fig. 7. Acceptance test for “Verify SMS code” scenario in SPL-AT Gherkin

A user in school security role can create new password with two different edit text and one button components on the creating new password page of mobile application. The critical requirement for this page is that the user should enter same password into the these edit text components. Since KidsBus™ system must ensure that given password is confirmed by the user. The test scenario outline shown in Fig.8 is created to test this feature.

Statistics about the files automatically generated for test execution are given in Table II. While three different children of the BasePage classes are created with using the rules, and also, three different children of the BaseTestNG classes are generated. Table II summarizes lines of code (LoC) generated for these classes as well as TestNG.xml file. As a result, three *Scenario Outlines* are covered within the generated mobile application test project. These acceptance tests can be extended with different combinations of the parameters in the Examples data table.

TABLE II. AUTOMATICALLY GENERATED FILES FOR TEST EXECUTION

Scenario	**Test Classes (LoC)**	**TestNG.xml (# of Lines)**
1. Get SMS code	62	15
2. Verify SMS code	98	17
3. Create New Password	145	26

```
Feature: Create New Password

Scenario Outline: Create new password scenario
Given @PAGE ReceiveVerificationCodeActivity is $OPENED
  And <username> is $ENTERED
  on @EDIT_TEXT usernameInput
  And @BUTTON loginButton is $CLICKED
  And @PAGE CommitVerificationCodeActivity is $OPENED
  And <passcode> is $ENTERED
  on @EDIT_TEXT activation_code
  And @BUTTON loginButton is $CLICKED again
  And @PAGE CreateNewPasswordActivity is $OPENED
When <new_password> is $ENTERED
  on @EDIT_TEXT new_password
  And <new_password_confirm> is $ENTERED
  on @EDIT_TEXT confirm_new_password
  And @BUTTON button_save_new_password is $CLICKED
Then @PAGE <result_page> is $OPENED
```

Fig. 8. Acceptance test for “Create new password” scenario in SPL-AT Gherkin

We explained *Scenario Outlines* for three pages of school security mobile application. The total number of *Scenario Outlines* is 14 with 152 lines in total for the whole school security mobile application. The total LoC for test classes automatically generated for the whole school security mobile application is 878. The length of TestNG.xml file is 176 lines.

V. RELATED WORK

In this paper, we focus on automatic generation of acceptance tests for features in SPLs. One research uses a decision model concept to maintain and generate acceptance test cases for SPLs [6], which saves space and effort as compared to conventional methods. There are three research that utilizes UML for automatic generation of acceptance tests in SPLs. One research utilizes both use case and sequence diagrams to compose behavioral test patterns [7]. Another one defines extensions and modifications of the Use Cases notation, which is called Product Line Use Cases (PLUCs) [8]. The last one proposes a functional test design method for SPLs called Customizable Activity diagrams, Decision tables and Test specifications, or CADeT, that defines a use case-based approach of creating reusable test specifications for a SPL [9].

Robot framework [10] is an open source automation framework for acceptance testing and acceptance test driven development. Different than our approach, which is feature-oriented by nature, users of Robot framework should define a feature-oriented language to represent feature-based scenarios. The proposed SPL-AT Gherkin can also be implemented using Robot framework.

VI. CONCLUSION

In this paper, we propose a feature-oriented testing approach for platform-based SPLs through a novel extension to Gherkin called SPL-AT Gherkin and a novel automatic test method generation technique based on TestNG framework. KidsBus™, which is a platform that manages the school bus transportation processes, is selected as case study. Three different pages in KidsBus™ School Security mobile application, which are getting SMS code, verifying the SMS code and creating new password, are tested with three different feature files written in SPL-AT Gherkin.

In the future, we are going to develop a test case management tool for the acceptance tests written in SPL-AT Gherkin. Moreover, we plan to connect the proposed approach with input contract testing based on Event Sequence Graphs [11] so that coverage-based test generation can be achieved for platform-based SPLs.

Acknowledgment

The authors would like to thank Tolgahan Oysal and Delta Smart Technologies (www.deltasmart.tech) for providing KidsBus™ to be used in the case study. This research is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the grant 117E884.

References

[1] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[2] D. S. de Souza, P. Vilain. "Selecting Agile Practices for Developing Software Product Lines", International Conference on Software Engineering & Knowledge Engineering (SEKE 2013), 220-225, 2013.

[3] Gherkin. https://docs.cucumber.io/gherkin/. Retrieved on April 14, 2019.

[4] Page Object Pattern. https://martinfowler.com/bliki/PageObject.html. Retrieved on April 12, 2019.

[5] Test Next Generation framework. (https://testng.org/doc/index.html). Retrieved on April 10, 2019.

[6] B. Geppert, J. Li, F. Rößler, and D. M. Weiss. "Towards generating acceptance tests for product lines". In International Conference on Software Reuse, pp.35-48. Springer, 2004.

[7] C. Nebut, S. Pickin, Y. Le Traon, and J. M. Jézéquel. "Automated requirements-based generation of test cases for product families". In 18th IEEE International Conference on Automated Software Engineering Proceedings, pp. 263-266, 2003.

[8] A. Bertolino, A. Fantechi, S. Gnesi, and G. Lami. "Product line use cases: Scenario-based specification and testing of requirements". In Software Product Lines, pp. 425-445, Springer, 2006.

[9] E. M. Olimpiew. "Model-based testing for software product lines", Doctoral dissertation, George Mason University, 2008.

[10] Robot Framework, https://robotframework.org/. Retrieved on May 15, 2019.

[11] T. Tuglular, F. Belli, and M. Linschulte. Input contract testing of graphical user interfaces. International Journal of Software Engineering and Knowledge Engineering, 26(02), pp.183-215, 2016.

Appendix-A

```
public class ThisIsIdentifierPage extends BasePage {
  public ThisIsIdentifierPage (AndroidDriver driver,
WebDriverWait wait) {
    super(driver, wait);
  }
  public void setThisIsEditTextIdentifier (String
delimetedParameter) {
    super.setText("thisIsEditTextIdentifier",
delimetedParameter);
  }
  public void clickThisIsButtonIdentifier () {
    super.click("thisIsButtonIdentifier");
  }
}


public class ThisIsTheTitleOfTheScenarioOutline extends
BaseTestNG {
  ThisIsIdentifierPage page = new ThisIsIdentifierPage
(driver, wait);
  @Test(priority = 0)
  public void Given_PAGE_this_is_identifier_is_OPENED
(String param) {
    // executed first
    // start appium here
  }
  @Parameters({"parameter_for_edit_text"})
  @Test(priority = 1)
  public void
When_parameter_for_edit_text_is_ENTERED_on_EDIT_
TEXT_this_is_edit_text_identifier (String param) {
    // executed second
    page.setThisIsEditTextIdentifier(param);
  }
  @Test(priority = 2)
  public void
And_BUTTON_this_is_button_identifier_is_PRESSED () {
    // executed third
    page.clickThisIsButtonIdentifier();
  }
  @Parameters({"parameter_for_page"})
  @Test(priority = 3)
  public void
Then_PAGE_parameter_for_page_is_OPENED (String
param) {
    // executed fourth
    assertEquals(param, ((AndroidDriver<MobileElement>)
driver).currentActivity());
  }
}
```

Appendix-B

```
<suite name="Suite">
  <test name="74129e81-7ce2-458b-8683-0a235978dc98">
    <parameter name="parameter_for_edit_text" value
="this_is_value_1_for_page">
</parameter>
    <parameter name="parameter_for_page"
value="this_is_value_1_for_page">
</parameter>
    <classes>
<class
name="Tests.ThisIsTheTitleOfTheScenarioOutline"></class
>
    </classes>
  </test>
  <test name="7f935cad-8d28-4dc4-8fc0-725286b83f87">
    <parameter name="parameter_for_edit_text" value
="this_is_value_2_for_page">
</parameter>
    <parameter name="parameter_for_page"
value="this_is_value_2_for_page">
</parameter>
    <classes>
<class
name="Tests.ThisIsTheTitleOfTheScenarioOutline"></class
>
    </classes>
  </test>
</suite>
```