Journal of
Information Systems Engineering
and Business Intelligence

Vol.6, No.1, April 2020
Available online at: http://e-journal.unair.ac.id/index.php/JISEBI

Tool for Generating Behavior-Driven Development Test-Cases

Indra Kharisma Raharjana[1)*], Fadel Harris[2)], Army Justitia[3)]

[1)2)3)] *Information Systems, Faculty of Science and Technology, Universitas Airlangga, Indonesia*
Kampus C Universitas Airlangga, Mulyorejo, Surabaya
[1)]indra.kharisma@fst.unair.ac.id, [2)]fadel.harris-13@fst.unair.ac.id, [3)]army-j@fst.unair.ac.id

Article history:

Received 2 March 2020
Revised 7 April 2020
Accepted 13 April 2020
Available online 28 April 2020

Keywords:

Acceptance Tests
Agile Behavior-Driven Development
Codeception
Test-Cases
Use Case Scenario
Software Development
Software Testing

Abstract

Background: Testing using Behavior-Driven Development (BDD) techniques is one of the practices of Agile software development. This technique composes a test-case based on a use case scenario, for web application acceptance tests.
Objective: In this study, we developed a tool to generate test case codes from BDD scenario definitions to help and facilitate practitioners to conduct testing.
Methods: The generated test case code is made according to the codeception framework format so that it can be directly executed by the tester. The procedure is performed as follows: map the correlation of the language used in BDD (gherkin language) and the code syntax of the test code in the codeception framework, designed the GUIs in such a way that users can easily transform the Use Case Scenario, built the tool so that it can generate test cases codes. Evaluation is done by gathering respondents; ask to run the application and gathering feedback from respondents.
Results: This tool can generate a codeception test-case file based on the BDD scenario. Generated test cases can be directly used on codeception tools. The results of the evaluation show that the tools can help entry-level programmers in developing automated tests.
Conclusion: The tool can help user especially entry-level programmers to generate BDD test-case and make easy for the users for testing the web applications.

I. INTRODUCTION

Behavior-Driven Development (BDD) is an agile software development best practice [1, 2]. BDD is basically a technique for building applications by determining acceptance tests as drivers for system development [3]. BDD focuses on the deliverable of system development behavior. It ensures that the developers and domain expert use the same language [4]. Solis et al [1] identified six characteristics of BDD: (1) Ubiquitous language; (2) Iterative decomposition process; (3) Plain text description with user story and scenario templates; (4) Automated acceptance testing with mapping rules; (5) Readable behavior oriented specification code; (6) Behavior driven at different phases. The same philosophy is also possessed by Test Driven Development [5, 6]. Testing techniques in agile software development usually apply Unit-testing and Test-Driven Development [7]. This type of technique is used to test low level units such as methods or components. Over time, automated testing that tests the combination of business logic and GUIs appears in BDD [8]. The acceptance testing can be a driver to create software that is suitable to user requirements and it could identify some problems [9]. The acceptance testing can be done by ordinary users. The testers could conduct acceptance testing and they could interact with the results directly.

Software testing can be done manually or automatically. The manual testing is done if all the activities are carried out by the testers, for example entering data and interacting with the object's page elements. While automatic testing has many advantages that are not possessed by manual testing, for example [10]: automatic testing can be done repeatedly in a short time, it is reusable because it can be done in various version [11], it can shorten the time when interacting with object elements, the automatic testing uses the script to be executed so that it does not need a lots of testers, the testing tools can find the location of the errors / bugs more specific, it can use the script so that automatic testing can use computer logic operations. Haugset et al [8] said that *Automated Acceptance Testing (*AAT) was more efficient in cost and also developers had more confidence using AAT.

However, automatic acceptance testing emphasizes the testers must run the testing tools correctly and prepare test-cases beforehand so it can be understood by the testing tools. A testing model is needed to help testers in

* Corresponding author

ISSN 2443-2555 (online) 2598-6333 (print)
doi: http://dx.doi.org/10.20473/jisebi.6.1.27-36

identifying the acceptance testing which they want to conduct. The Behavior-Driven Development (BDD) model can be a solution to this problem. Based on the research [1], one of the characteristics of the BDD model is that it can automate the acceptance testing with mapping (mapping). The acceptance testing using the BDD model can identify the specifications of the system formed in the scenario; the scenario will verify the interaction behavior that occurs in the test object.

Currently, there are many tools that can automatically perform acceptance testing, such as *codeception* (https://*codeception*.com/). *Codeception* can implement user acceptance testing with a viewpoint of user desires and conditions using the BDD development model. *Codeception* only works on testing web-based software. Lazar [4] exploited Eclipse-based development tools to develop *fuml* components using BDD approach. Carrera et.al [12] developed BEhavioral Agent Simple Testing (BEAST) framework which was able to produce a test case based on BDD Scenario. His framework can run under Java Agent Development Framework (JADE), a software framework implemented in the Java language. Tavares et.al [13] chose Python Language to implement BDD. The result using Python Language had minimized error rate in functional and unit level.

Considering the previous studies, we attempt to develop BDD-based automated acceptance testing using *codeception*. One of the advantages of the *codeception* tool is that there is a BDD model test-case, it is capable of directly testing and documenting how the test runs. If there is an error during the test, the *codeception* can provide information to the user where the error is. to the best of our knowledge, no tools are publicly available for automating the process and this is may be the first one of its type.

The need for using the tool *codeception* is a special BDD model test-case that requires writing according to the *codeception* tool format. This is a problem if the testers who have BDD model testing scenarios must adjust the scenario with the format *codeception* tool. The solution to this problem is to use automation tools, the testing scenarios owned by testers can be transformed according to the format *codeception* tool. Therefore, this study will build tools that can transform BDD modeled test scenarios into *codeception* formatted test-cases, so tools will be available to create BDD models, test scenarios owned by testers can be transformed automatically, and time taken to use fewer *codeception* tools. The purpose of this study is to build tools that can automatically create Behavior-Driven Development test-cases for web-based software.

II. LITERATURE REVIEW

A. User Stories and Scenarios in BDD

User stories are multi-purpose natural language, which can be used to explain user desire, product descriptions, planning items, tokens for a conversation, or mechanism for deferring conversation [14, 15]. User stories greatly ease communication with end users who have no background in IT knowledge. User stories are semi-structure languages [16], with a simple template shown in Fig. 1. A simple example of user stories is as follows [17] *"As a Facebook user, I want to be able to login into my account, so that I can share my photos"*.

As a <type of user>, I want <some goals> so that <some reasons>

Fig. 1 A Simple User Story Template

Despite it provides convenience in understanding system features, user stories still have some obstacles in their implementation. Natural language is difficult to understand and process directly by the computer, so for some processes that should be automated it must be done manually (for example: making a test case). In addition, the process of validation and verification of Existing user stories often creates conflicts between users and developers. This is caused by the ambiguity caused using user stories and the scope described in the user story is unclear. To overcome this problem, the user stories can be detailed using acceptance criteria [18].

North [19] and Cohn [20] introduced user stories template with a set of scenarios that describe the acceptance criteria. The user story template and an example can be seen in Fig. 2 and Fig.3, respectively. Scenarios describe how system features must behave in certain circumstances and an event occurs. The results of scenarios are actions that change the state of the system or produce system output. The attributes of scenario consist of the **Title**, **Given** is an initial condition before the scenario is run (precondition), **When** is the condition when the user (As a) performs an action, and **Then** is the response of the system. This user story scenario uses the standard Gherkin language [21]. Gherkin language is a domain specific language create especially for behaviors descriptions [22].

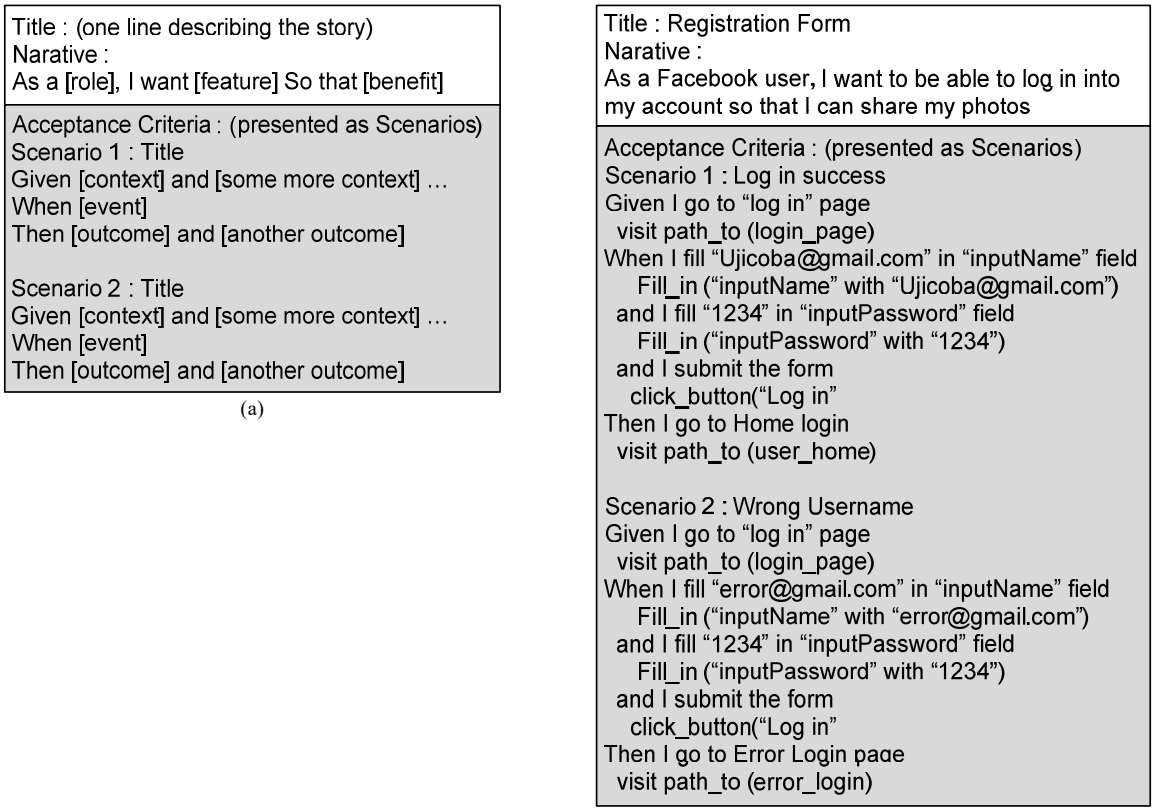


Fig. 2 Example of User Story (a) Template of User Story (Introduced by [19] and [20]) (b) An example of User Stories with 2 Scenarios

B. Codeception

Codeception is a versatile, feature-rich testing framework for PHP web-based applications. It provides automated testing tools based on BDD scenarios to help developers and testers focus on testing. *Codeception* support acceptance testing, functional testing, and unit testing [23]. In acceptance testing, *codeception* provides several functions. These functions are used to document the testing steps that will occur, so that the *codeception* is able to understand the test desired by the user [24]. Table 1 presents these functions and explains their use in acceptance testing. Like other tools, *codeception* also requires several components to be run. These components are Composer and Terminal (on Mac) or Command line (on Windows) [25].

TABLE 1
CODECEPTION FUNCTION FOR ACCEPTANCE TESTING

Codeception function	Explanation
I->wantTo ('Input');	Desired scenario
I->amOnPage ('Input');	Precondition
I->click('#Element');	When the user presses a button
I->fillField('#Element','Input');	When the user fills in a column
I->selectOption('#Element','Input');	When the user selects an option
I->seeInCurrentUrl('Input');	Validate the current user's page
I->wait('Input');	Wait for a few seconds
I->see('Input');	Validation of visible parts
I->dontSee('Input');	Validation of the invisible parts
I->seeElement('#Element');	Validation on elements

III. METHODS

This study produced a tool that can generate a Behavior-Driven Model test-case on a web-based application. Identification of inputs, processes, and outputs are needed to be able to build this tool. The input in this tool is a BDD scenario. The process carried out is inputting the BDD scenario transformed into a *codeception* test-cases. The

output of this tool is a *codeception* test-case file that is ready to run. Illustration of system architecture can be seen in Fig. 4.

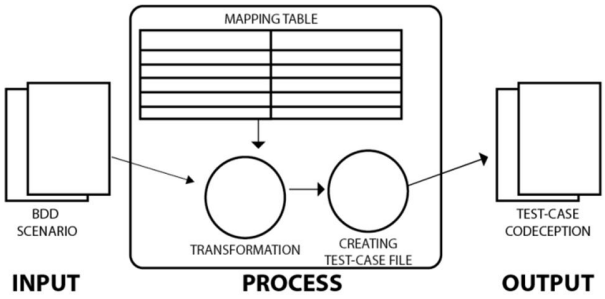


Fig. 3 System Architecture

There are five steps carried out in the study. The first step is the creation of a mapping table that contains guidance for transforming the BDD scenario into a test-case *codeception*. The second step is designing the process that determines the method for transforming BDD scenarios into *codeception* test-case. The third fourth step is designing the output which contains how to create a new file according to the *codeception* format. Tool evaluation is done by collecting questionnaires and responses from users. The detailed research procedure can be seen in Fig. 4.

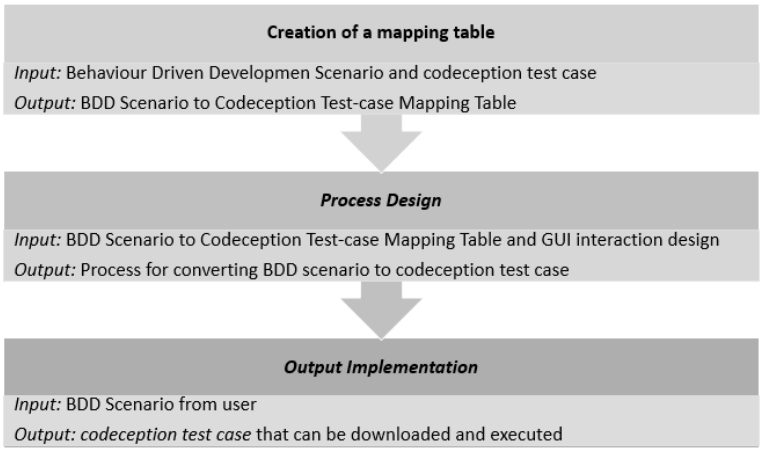


Fig. 4 Research procedure

A. Analysis and Table Mapping Design

There were two sets that must be made; those are the set of BDD scenarios and the set of *codeception* test-case. The element of the set of *codeception* test-case was the *codeception* function in the acceptance test, whereas the element of the BDD scenario set is the translation of the *codeception* test-case set of elements from the user point of view when performing the activity. BDD scenario elements were divided into three parts: the precondition, the **When I** section, and the **Then I** section.

In the set of test-case *codeception*, there is a function *I->wantTo ('Input'), I->amOnPage('Input')* which is a precondition in the BDD scenario. Then there is *I->click('#Element'), I->fillField('#Element','Input'), I->selectOption('#Element','Input')* which is part of *When I* in the BDD scenario. In the part *Then I* there are *I->seeInCurrentUrl('Input'), I->wait('Input'), I->see('Input'), I->dontSee('Input'), dan I->seeElement('#Element')*. BDD Scenario to *Codeception* test-case mapping tabel shown in Fig. 5.

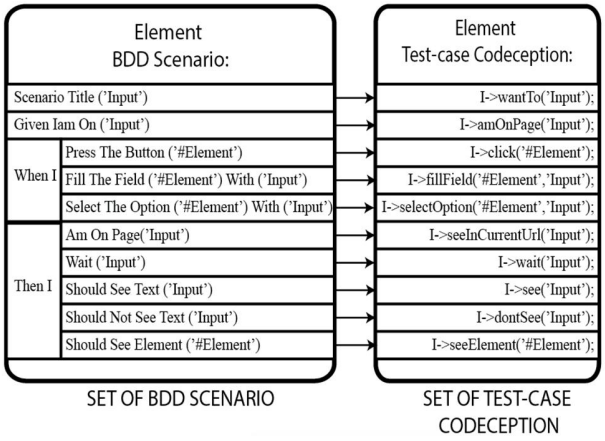


Fig. 5 BDD Scenario to Codeception Test-case Mapping Table

B. Process Design

The design process discusses how the process of changing each inputted BDD scenario into a *codeception* test-case file. There are two processes that will be shaped, namely the transformation process and the process of making a test-case file.

The transformation process begins when the user finishes inputting the entire scenario. The system reads each step of the input scenario and then converts it to a *codeception* function according to the mapping table (See Fig. 2). For example, if at the scenario stage there is *When I Press The Button ('Login')* it will be changed to *I->click('Login');*. The transformation process is stopped when there is no longer a BDD scenario to be transformed. After each BDD scenario is successfully changed based on the mapping table, the system then creates a file with the same title as the scenario name using '.php' as a file extension.

C. Output Implementation

The output of this tool is a test-case file that ready to be used in *codeception* tools. The title of scenario in the input scenario is used as the filename, then followed by the word 'Cept' because the *codeception* tool only runs with that format. The contents of the test-case file are the result of the transformation of the inputted scenario. The extension of the file that is formed is '.php' because the acceptance testing of the *codeception* tool can only run files with that extension.

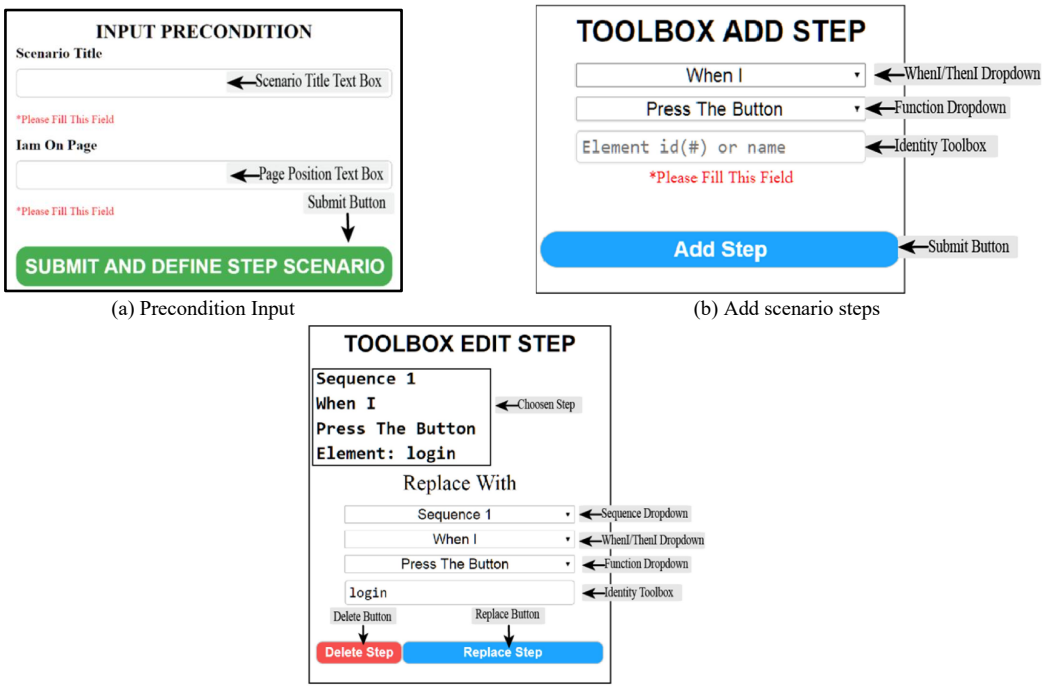
IV. RESULTS

A. GUI Design

In using tools, users interact with the system through the graphical user interface. Interaction design is arranged so that users can easily use the tool. The entire input scenario process will use the GUI form. Entering the BDD scenario begins with the precondition of the BDD scenario (Fig. 6a). In accordance with the precondition attribute of the BDD scenario, the input form must consist of 1 text box to fill the scenario title, 1 text box to fill the user's starting page position (*Given I Am On*), and 1 submits button to complete the preconditioning process.

After preconditions, the BDD scenario must have a scenario step (Fig. 6b). Filling out the scenario steps is done by adding one by one the scenario steps. The scenario step in the BDD scenario begins with the choice of *When I* or *Then I*. The functions contained in the mapping table are according to *When I* or *Then I* choices and followed by their identities. To fill in the scenario steps, the step input screen scenario must consist of 1 dropdown to select *When I* or *Then I*, 1 dropdown to select the choice of functions contained in the mapping table, 1 text box to fill in the identity of the required function, along with 1 button to submit to add to the scenario step.

When the user has finished creating the step-by-step scenario and wants to change or delete the scenario steps, Users must be able to make these changes. Changing or deleting scenarios can occur when there has been at least one step scenario. In the GUI there is information on the steps of the scenario that you want to change, which part you want to change (Fig. 6c). Changeable parts are the sequence of steps, the choice of **When I / Then I**, the choice of functions, or changing the identity of the function.



(a) Precondition Input (b) Add scenario steps

(c) Change scenario steps

Fig. 6 Interaction Design

In order to use this tool, Software under tests and the BDD scenario must be prepared in advance. Software Under test is in form of web application. The web application will be tested based on the testing scenario, which is made in the form of a BDD scenario. Fig. 7 shows an example of a BDD scenario. BDD scenario can be obtained from the transformation of the use case scenario that was created at the software analysis and design phase.

In order to generate *codeception* test code, the information from the BDD scenario is entered to the tool. scenario name (*scenario:login success*) and precondition (*Given i am on: login.php*) are inputed in tool, as shown in Fig. 8a. every step in the BDD scenario is entered to the tool, as shown in Fig. 8b. Fig. 8c shows the results of all the scenario steps that have been entered into the tool.

```
SCENARIO : LOGIN SUCCESS
GIVEN IAM ON: LOGIN.PHP
WHEN I : Fill the field "Username" with "Adam"
AND I :  Fill the field "Password" with "Adam123"
AND I :  Click the button "LOGIN"
THEN I :  Am on page "home.php"
```

Fig. 7 Example of a BDD scenario

After all the scenarios are entered into the tool, users can generate a test case. The result is a test case code as shown in Fig. 9. This code can be downloaded and directly executed using a *codeception* framework to test the system (seen in Fig. 10)

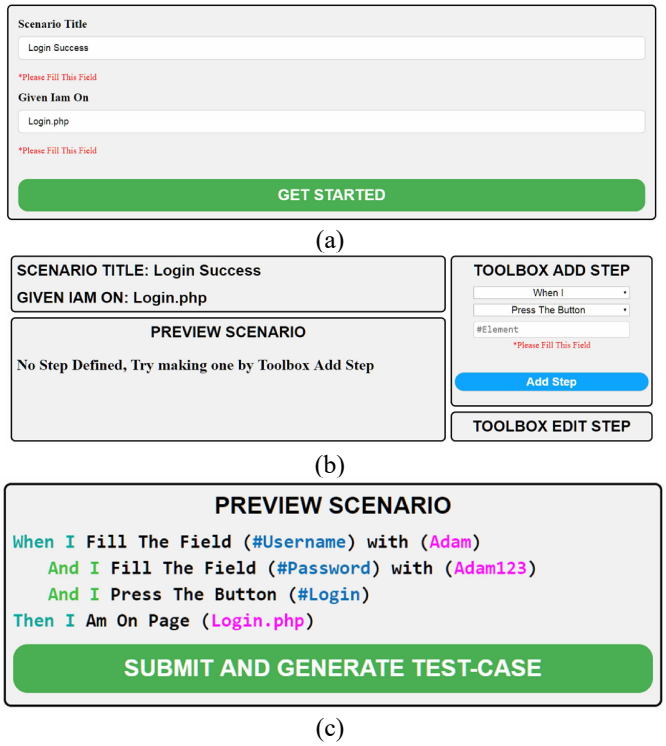


Fig. 8 GUI of the System

B. Evaluation

The tool has been tested by inputing several BDD scenarios into the system and producing output as desired. The tool is able to transform the BDD scenario into a test-case file in the *codeception* format. This test-case file can be executed directly within *codeception* framework.

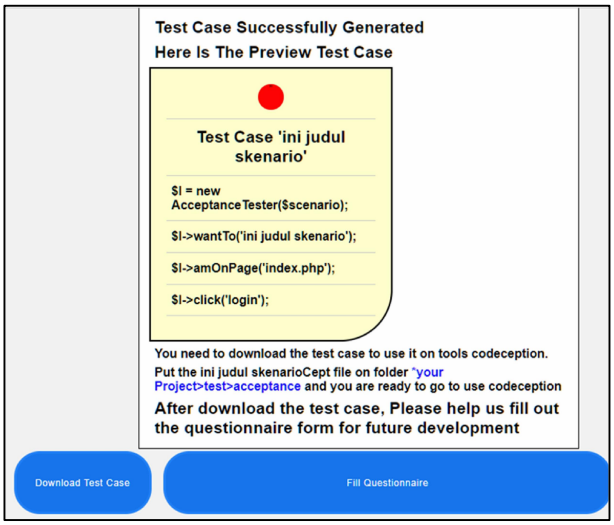


Fig. 9 Test Code Output

```php
<?php
$I = new AcceptanceTester($scenario);
$I->wantTo('Login Facebook');
$I->amOnPage('/login');
$I->fillField('inputName','Ujicoba@gmail.com');
$I->fillField('inputPassword','1234');
$I->click('login');
$I->seeInCurrentUrl('/halaman utama profil');
```

Fig. 10 Example of Test code generated

The evaluation of the tool was done by gathering respondents and distributing questionnaires. We posted an online questionnaire and link to our tools, so respondent can run the tool and evaluated it. Most of the respondents are entry level programmers. Respondents were asked to prepare test case scenario as input of the system.

After respondent tries out the system, then they fill up the questionnaire based on their experience (seen in Fig. 11). Respondents' responses were collected to verify the functionality, accuracy, and satisfaction of the tool.

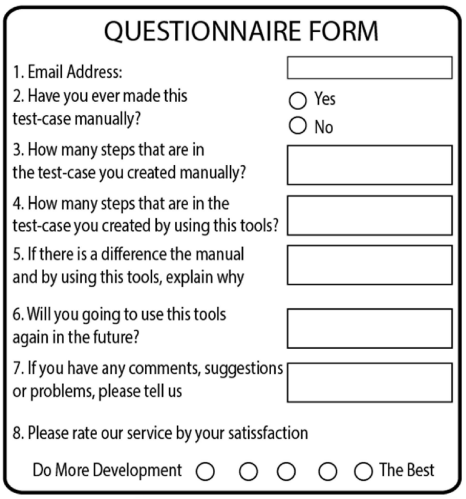


Fig. 11 Questionnaire Form

There were 19 respondents, 6 respondents had made test cases manually before, while 13 respondents who first made a test case using this tool. All respondents agree that this tool can convert BDD scenarios into test-cases based on the *codeception* format and can be used directly within the framework. It was concluded that this tool has fulfilled the required functionality. Most of respondents were satisfied when they were given the opportunity to test the tool, 18 respondents said they wanted to use this tool again.

V. Discussion

In the perspective of agile software development, BDD is compiled based on user stories and scenarios. User stories and scenarios are arranged according to the normal flow, alternatives and exceptions of user stories. With the automated test case generator tool, it is expected to accelerate the process of software development especially the process of decomposing user stories into user stories and scenarios. User stories can have 3 levels of desire, namely goal, task, capability. In software development practices, the goal level must be broken down into more atomic user stories that have the type of task or capability. By implementing BDD in software development, automatically the developer must change the user story to the type of task and capability. This can improve the quality of the user story in the requirements engineering phase, especially in defining user stories as requirements artifacts.

The development of tools for generating BDD test cases is still a one-time creation of user stories and scenarios. In a sense if there are changes of scenarios after the test case has been generalized, the tester must start from scratch to generate the test case. Respondents also gave suggestions to improve the UI / UX of the tool so that it was easier and not confusing respondents.

From the results of the evaluation, we realized that the tools we develop can helped entry level programmers in developing automated tests. but it needs some improvements to be able to use this tool at the production level and be used by professional software development, such as improve the tool in UI/UX, able to re-read and change test cases

from test cases that were successfully generated before, adding an automated test repository system as proposed by Rahman and Gao [26], or integrating with tools to help in generating source code [27].

VI. CONCLUSIONS

The results support main contributions of this study, users (entry level programmers) can make it easier to do Behavior-Driven Development testing using this tool. In this study, a tool has been developed to help users create *codeception* test code simply by entering information from a use case scenario. This tool uses the standard Gherkin language to be entered by users into the system, based on the use case scenario. The output is a test code that is ready to be executed by the user using *codeception* framework. For future work, we will try to combine the requirements model with the test case generator. Especially, user story and user story scenario can be used to generate automated system testing.

REFERENCES

[1] C. Solís and X. Wang, "A Study of the Characteristics of Behaviour Driven Development," in *Conference on Software Engineering and Advanced Applications (SEAA)*, 2011.

[2] A. Aitken and V. Ilango, "A Comparative Analysis of Traditional Software Engineering and Agile Software Development," in *2013 46th Hawaii International Conference on System Sciences*, 2013.

[3] M. Soeken, R. Wille and R. Drechsler, "Assisted Behavior Driven Development Using Natural Language Processing," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Berlin, 2012.

[4] I. Lazar, B. Parv and S. Motogna, "Behaviour-Driven Development of Foundational UML Components," *Electronic Notes in Theoritical Computer Science,* vol. 264, pp. 91-105, 2010.

[5] B. George and L. Williams, "A structured experiment of test-driven development," *Information and Software Technology,* vol. 46, p. 337–342, 2004.

[6] D. Janzen and H. Saiedian, "Test-Driven Development: Concepts, Taxonomy,and Future Direction," *Computer,* pp. 43-50, 2005.

[7] H. Erdogmus, M. Morisio and M. Torchiano, "On the effectiveness of the test-first approach to programming," *IEEE Transactions on Software Engineering,* vol. 31, no. 3, 2005.

[8] B. Haugset and G. K. Hanssen, "Automated Acceptance Testing: a Literature Review and an Industrial Case Study," in *Agile 2008 Conference*, 2008.

[9] F. D. Davis and V. Venkatesh, "Toward Preprototype User Acceptance Testing of New Information Systems: Implications for Software Project Management," *IEEE Transactions on Engineering management,* vol. 51, no. 1, pp. 31-46, 2004.

[10] P. Rathi and V. Mehra, "Analysis of Automation and Manual Testing Using Software Testing Tool," *International Journal of Innovations & Advancement in Computer Science,* vol. 4, pp. 709-713, 2015.

[11] M. Rahman and J. Gao, "A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development," in *IEEE Symposium on Service-Oriented System Engineering*, San Francisco, 2015.

[12] A. Carrera, C. A. Iglesias and M. Garijo, "BEAST methodology: An agile testing methodology for multi-agent systems based on behaviour driven development," *Information System Frontiers,* vol. 16, no. 2, pp. 169-182, 2014.

[13] H. L. Tavares, G. G. Rezende, V. M. Santos, R. S. Manhães and R. A. Carvalho, "A tool stack for implementing Behaviour-Driven Development in Python Language," Instituto Federal Fluminense, Brazil, 2010.

[14] J. Patton and P. Economy, User Story Mapping: Discover the Whole Story, Build the Right Product, United States of America: O' Reilly Media Inc., 2014.

[15] I. K. Raharjana, D. Siahaan and C. Fatichah, "User Story Extraction from Online News for Software Requirements Elicitation: A Conceptual Model," in *The 16th International Join Conference on Computer Science and Software Engineering (JCSSE19)*, Pattaya,Thailand, 2019.

[16] I. K. Raharjana, Pengembangan Sistem Informasi Menggunakan Metodologi Agile, Yogyakarta: depublish, 2017.

[17] G. Lucassen, F. Dalpiaz and J. M. E. M. van der Werf, "The use and effectiveness of user stories in practice," in *International working conference on requirements engineering: Foundation for software quality*, Cham, 2016.

[18] T. R. Silva, J.-L. Hak and M. A. Winckler, "An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development," *Complex Systems Informatics and Modelling Quarterly,* pp. 81-107, 2016.

[19] D. North, "What's in a story?," 2016. [Online]. Available: https://dannorth.net/whats-in-a-story/. [Accessed 29 February 2020].

[20] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley Professional, 2004.

[21] blog.lepine.pro, "Behat Cheat Sheet," 6 Mar 2012. [Online]. Available: http://blog.lepine.pro/php/ressources-tutos-php/cheat-sheet-behat. [Accessed 3 Mar 2020].

[22] I. Papadopoulos, D. Kogias, C. Patrikakis and C.-. C. Marinou, "Enhancing the Student's Logical Thinking with Gherkin Language," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, Athens, Greece, 2017.

[23] R. Dāsa, "Test-Driven Development," *Learn CakePHP,* pp. 18-21, 2016.

[24] C. Team, "Codeception," 2011. [Online]. Available: https://codeception.com/docs/03-AcceptanceTests. [Accessed 29 February 2020].

[25] Sugiarto, B. Nugroho and A. D. Putri, "Penggunaan Metode Acceptance Test Driven Development pada Proses Acceptance Test Menggunakan Codeception," *Scan,* pp. 45-50, 2019.

[26] M. Rahman and J. Gao, "A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development," in *2015 IEEE Symposium on Service-Oriented System Engineering*, San Francisco Bay, CA, USA, 2015.

[27] M. Landhaußer and A. Genaid, "Connecting User Stories and code for test development," in *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*, Zurich, Switzerland, 2012.