A Connectivity Improvement Method for Behavior Driven Acceptance Tests

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

Nazım Umut Ekici
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkey
mail: nazimekici@iyte.edu.tr

***Abstract*—Behavior Driven Acceptance Tests (BDATs) are not necessarily written in a semantic flow. Sometimes they are written in an ad-hoc manner, and some other times they are grouped by features or requirements. Connecting BDATs for faster test execution may prevent reset or set operations in test environments. Moreover, if BDATs cannot be connected, that may mean missing BDATs. Therefore, better-connected BDATs result in better implementation and testing. This work proposes a method for improving the connectivity of BDATs utilizing natural language processing techniques and a graph model-based test generation technique called Event Sequence Graphs (ESGs). For the connection of BDATs, we utilize the technique called elimination of tags by combination in ESGs introduced in our previous work. The proposed method here improves the connectivity of existing behavior-driven acceptance test suites. It is validated through two non-trivial examples. The results demonstrate the feasibility of the proposed method.**

Keywords-model-based testing; event sequence graphs; behavior driven acceptance tests; Gherkin.

I. INTRODUCTION

The proposed method in this paper improves the connectivity of behavior-driven acceptance test suites developed using the method presented in [1]. Behavior Driven Development (BDD) is focused on defining fine-grained specifications of the behavior of the targeted system [2]. In BDD, tests are clearly written using a specific ubiquitous language, such as Gherkin [3]. For developing Behavior Driven Acceptance Tests (BDATs), there are environments like Cucumber [3], which forces testers to use a test template using Gherkin language and environments like Gauge [4], which does not impose any language. The scope of this study is BDATs developed in Gherkin.

Although Gherkin and its scenario template helps test designers in writing test scenarios, there is no guidance on the connectivity of test scenarios. All public Github repositories are searched for files with the extension ".feature". Github does not report the number of unique repositories that match a given query. Instead, it reports that there are over 2M unique files that match the query at the time of reporting. Github also limits the query results to the top 1000 files, most of which are hosted in the same repositories (i.e., 1000 file results are not from 1000 unique repositories). By executing the same query at different times, we collected 1314 unique repositories with Gherkin scenarios. The largest repository has 1041 feature files. Our search results can be found at https://github.com/esg4aspl/Gherkin-Scenario-Collection-and-Analysis/blob/main/README.md. We analyzed 5% of these 1314 repositories manually and did not find any work (i.e., explanation, code) related to connectivity of Gherkin scenarios. This work addresses this problem and provides a method for improvement of connectivity of BDATs.

The method proposed here utilizes natural language processing (NLP) techniques and a graph model-based test generation technique. Therefore, we borrow the connectivity definition from the theory of directed graphs. As a solution, we utilize semantic similarity measure to tag BDATs, then transform tagged BDATs into formal graph test models, and finally connect them through elimination by composition method introduced in [1]. So, if there are unconnected BDATs, or Gherkin test scenarios, the proposed method warns test designer to improve existing BDATs by adding new BDATs.

The proposed method assumes that clauses written in Gherkin can be represented by events. In that case, an event-based formal model would fit better to BDATs. Therefore, we propose the use of Event Sequence Graphs (ESGs) for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags. This is one of the novelties presented in this paper. Another novelty presented here is the process of finding missing BDATs. To find missing BDATs, the proposed method follows elimination of tags by combination. After the missing BDATs are completed, an ESG without any tags is obtained. The proposed method is explained with a running example in Section III. For evaluation, a BDAT test suite is selected from Github™ and the proposed method is applied to this test suite. The results are shared in Section IV.

This paper makes the following main contributions:

(i) Method: The proposed method creates a corpus from exiting Gherkin statements and tries to match end of a test scenario with the beginning of another test scenario through semantic similarity. A unique tag is automatically generated for the matched statements. For the unmatched statements unique tags are also generated with a table entry of close statements. Then tagged BDATs are transformed to tagged ESGs and combined by utilizing the elimination by composition method introduced in [1]. Analysis of the resulting ESG or ESGs reveals improvement in the connectivity of BDATs.

(ii) Tool: We developed a tool that implements the method explained in (i) and shared in a public repository.

The manuscript is organized as follows: In the next section, the fundamentals of the concepts used in this research are given along with examples and figures. The proposed method is explained in Section III using a running example and Section VI presents the software tools developed and used in this research. Section V gives an evaluation of the proposed method along with a discussion in Section VI. Section VII sets out the threats to validity. Section VIII outlines related work, and the last section concludes the paper.

II. FUNDAMENTALS

A. Gherkin

Gherkin uses a set of special keywords to give structure and meaning to executable specifications [3]. It provides the behavior definitions of the intended software not only to product owners and business analysts, but also to developers and testers [5]. Gherkin is a line-oriented language in terms of structure and each line must be divided by the Gherkin keyword except feature and scenario descriptions [3]. In this paper, some of the Gherkin keywords; namely *Feature*, *Scenario*, *Given*, *When*, *And*, *Then*, are utilized. Throughout the paper, the terms Gherkin scenario, scenario, and BDAT are used interchangeably.

Tests should be independent of each other so that they can be run in any order or even in parallel. This principle is also applied in developing BDATs. So, each BDAT should be run manually or automatically independent of other BDATs. However, they should also be composable so that it will be possible to execute a BDAT after a related one.

B. Natural Language Processing

Cosine similarity is a method for measuring similarity between two vectors [6]. By converting text documents to vectors, cosine similarity is widely used to assess the similarity between documents. Term Frequency/Inverse Document Frequency (TF-IDF) is used to convert text documents to vectors [6]. Given a corpus and a document from the corpus, for each word in the document, TF-IDF uses the frequency of the word in the document (its significance for the document) and the corpus (its informativeness for the corpus). By normalizing TF with DF, TF-IDF outputs a relative significance of each word in the document with respect to the corpus. TF-IDF and cosine similarity are commonly used together to assess the similarity of arbitrary documents in the context of the corpus [7], [8].

Text pre-processing is an essential part of NLP, which aims to improve any further processing [9], [10]. Two common types of pre-processing are stop word removal and stemming. Stop words are frequent words in the language, which have little informativeness (e.g., the, a, an for the English language) but can affect the output. Their removal also helps reduce the size of the corpus. Stemming is applied to words to strip them from any modifiers and transform them to their root form. For instance, withdrawal, withdrawing, and withdraws can all be stemmed from the word withdraw. Stemming aids in identifying semantic similarities out of the syntactic context.

C. Event Sequence Graphs

A model of the system, which requires the understanding of its abstraction, helps in testing its behavior. A formal specification approach that distinguishes between legal and illegal situations is necessary for acceptance testing. These requirements are satisfied by event sequence graphs [11].

Differing from the notion of finite-state automata, inputs and states are merged in ESG, hence they are turned into "events" to facilitate the understanding and checking the external behavior of the system. Thus, vertices of the ESG represent events as externally observable phenomena, e.g., a user action or a system response. Directed edges connecting two events define allowed sequences among these events [11]. Definitions from 1 to 3 and related examples and explanations along with Figure 1 are taken exactly as presented in [12]-[15].

Definition 1. An *event sequence graph ESG = (V, E, Ξ, Γ)* is a directed graph where $V \neq \emptyset$ is a finite set of vertices (nodes), $E \subseteq V \times V$ is a finite set of arcs (edges), $\Xi,\Gamma \subseteq V$ are finite sets of distinguished vertices with $\xi \in \Xi$, and $\gamma \in \Gamma$, called entry nodes and exit nodes, respectively, wherein $\forall v \in V$ there is at least one sequence of vertices $\langle \xi, v_0, \ldots, v_k \rangle$ from each $\xi \in \Xi$ to $v_k = v$ and one sequence of vertices $\langle v_0, \ldots, v_k, \gamma \rangle$ from $v_0 = v$ to each $\gamma \in \Gamma$ with $(v_i, v_{i+1}) \in E$, for $i = 0, \ldots, k\text{-}1$ and $v \neq \xi, \gamma$.

To mark the entry and exit of an ESG, all $\xi \in \Xi$ are preceded by a pseudo vertex '[' $\notin V$ and all $\gamma \in \Gamma$ are followed by another pseudo vertex ']' $\notin V$. The semantics of an ESG are as follows. Any $v \in V$ represents an event. For two events $v, v' \in V$, the event v' must be enabled after the execution of v iff $(v, v') \in E$. The operations on identifiable components of the GUI are controlled and/or perceived by input/output devices, i.e., elements of windows, buttons, lists, checkboxes, etc. Thus, an event can be a user input or a system response; both are elements of V and lead interactively to a succession of user inputs and expected desirable system outputs.

Example 1. For the ESG given in Figure 1: $V=\{a,b,c\}$, $\Xi=\{a\}$, $\Gamma=\{b\}$, and $E = \{(a,b), (a,c),(b,c),(c,b))\}$. Note that arcs from pseudo vertex [and to pseudo vertex] are not included in E.

Furthermore, α(initial) and ω(end) are functions to determine the initial vertex and end vertex of an ES, e.g., for ES= $(v_0, \ldots, v_k)$ initial vertex and end vertex are $\alpha(ES)=v_0$, $\omega(ES)=v_k$, respectively. For a vertex $v \in V$, $N^+(v)$ denotes the set of all *successors* of v, and $N^-(v)$ denotes the set of all *predecessors* of v. Note that $N^-(v)$ is empty for an entry $\xi \in \Xi$ and $N^+(v)$ is empty for an exit $\gamma \in \Gamma$.

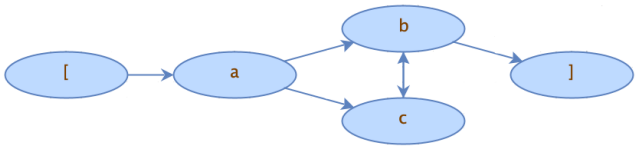

Figure 1. An ESG with a as entry and b as exit and pseudo vertices '[' and ']'.

Definition 2. Let V, E be defined as in Definition 1. Then, any sequence of vertices $\langle v_0, \ldots, v_k \rangle$ is called an *event sequence* (*ES*) iff $(v_i, v_{i+1}) \in E$, for $i=0, \ldots, k-1$.

The function l(*length*) of an ES determines the number of its vertices. If l(ES)=1 then ES=(v_i) is an ES of length 1. Note that the pseudo vertices [and] are not considered in generating any ESs. Neither are they included in ESs nor considered to determine the initial vertex, end vertex, and length of the ESs. An ES = $\langle v_i, v_k \rangle$ of length 2 is called an *event pair* (EP).

Definition 3. An *ES* is a *complete ES* (or, it is called a *complete event sequence*, *CES*), if α(ES)=$\xi \in \Xi$ is an entry and ω(ES)=$\gamma \in \Gamma$ is an exit.

A CES may not invoke interim system responses during user-system interaction. If it does not, that means that it consists of consecutive user inputs and only a final system response. CESs represent walks from the entry of the ESG to its exit, realized by the form (initial) user inputs → (interim) system responses → ⋯ (interim) user inputs → (interim) system responses → ⋯ → (final) system response.

ESGs are hierarchical models enabling sub-ESGs, or sub-models, which are also ESGs. A hierarchical ESG can be refined or flattened to one layer. Please see [12]-[15] for further details. Therefore, we can say that ESGs support and manage large models by following the divide-and-conquer approach in computer science.

D. Connectivity in Directed Graphs

Two vertices u and v in a graph G are connected if $u = v$, or $u \neq v$ and a u-v path exists in G [16]. A graph G is connected if every two vertices of G are connected; otherwise, G is disconnected [16]. The ESG obtained after graph transformation of BDATs and their composition might be a disconnected directed graph. By improving the connectivity of this graph, we would like to make it a connected ESG, so that test sequences can be automatically generated and reset operations are minimized.

III. Proposed Method

The proposed method improves connectivity of a BDAT test suite by NLP analysis, graph-based modeling, and model composition. The proposed method not only improves connectivity but also enables coverage-based test sequence generation by ESGs.

With the assumption that Gherkin clauses can be represented by events, the proposed method suggests the use of ESGs for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags. This is explained first in this section. Then, how BDATs are combined using tagged ESGs is presented. After that, elimination of tags by combination process that is used to find missing BDATs is outlined. This section concludes with an example where all BDATs, i.e., original, missing, and additional BDATs, are composed into one ESG without any tags.

A. Extension of BDATs with tags

Best practice for Gherkin scenarios is to describe behavior rather than functionality. A behavior driven acceptance test is a specification of the behavior of the system, which verifies the interactions of the objects rather than their states [17]. A scenario that makes up a BDAT is composed of several steps. A step is an abstraction that represents one of the elements in a scenario which are: contexts, events, and actions [2]. So, a Gherkin scenario template is as follows:

```
Given context
When event
Then action
```

Contexts, events, and actions can be represented by events. A context, or state, is formed after a sequence of events. For instance, the line Given I am on the homepage in a scenario indicates that the context is being on the homepage and the user can reach the homepage by a sequence of events. So, we can say that a context is the result of a sequence of events. Sometimes, the sequence of events may be empty. An action is an event or results in an event depending on your standpoint. For instance, the line Then product list is displayed in a scenario is the action of the software, but for the user it is an event. We conclude that all Gherkin clauses are either events or a result of an or a series of events and therefore, we claim that all Gherkin clauses can be expressed as events.

Algorithm 1 defines the steps for extracting semantic relations between step definitions by utilizing NLP techniques. Given a set of Gherkin scenarios, a corpus is constructed by aggregating all step definitions from all scenarios. Punctuation and stop words are removed from the corpus to highlight more important words. Furthermore, words are stemmed to reduce them to their root form. After the conversions, pairwise cosine similarity is calculated for all items in the corpus by using the TF-IDF transformation [6]. For every Given and Then step definition, their similarity scores with every other Given and Then step definition are collected to a list. The list is sorted in the descending order of similarity score. The resulting collection lists best matching step definitions for each step definition. Output of the algorithm can be interpreted as a list of match suggestions for each step definition.

Algorithm 1 is applied to the running example. Number of correct matches present in each list length is also presented in Table I up to a list length of 5. Table I shows that, out of 17 possible matches, 12 of them are correctly identified as the best match by Algorithm 1. All possible

matches are identified within the first 5 suggestions by the algorithm.

Algorithm 1. Match Gherkin Then Statement and Given Statement Pairs

```
Input: Scenarios
Corpus ← { }
For each scenario in Scenarios:
    Corpus = Corpus U scenario.stepDefinitions
Endfor
Corpus.removePunctuation()
Corpus.removeStopWords()
SimilarityScores = Corpus.calculatePairwiseSemanticSimilarity()
sortedMatchesForStepDefinitions = { }
For each stepDefinition in Corpus:
    Scores = SimilarityScores.getScoresForStepDefinition(stepDefinition)
    Scores.sortDescending()
    sortedMatchesForStepDefinitions[stepDefinition] = Scores
Endfor
Output: sortedMatchesForStepDefinitions
```

TABLE I. CORRECT TAG MATCH COUNTS IN A GIVEN LIST LENGTH FOR THE RUNNING EXAMPLE

Tagged step definition count	*List length 1*	*List length 2*	*List length 3*	*List length 4*	*List length 5*
17	12	14	15	15	17

As an example of Algorithm 1's results, consider the two Given step definitions from the running example below. Step definitions are best match for each other according to Algorithm 1. Both Given step definitions describe the same state in the program execution. Therefore, the match is considered to be correct.

Scenario: srch01- Do a valid search with a single keyword
Given I am on the homepage to do a single keyword search
Scenario: srch02- Do a valid search with multiple keyword
Given I am on the homepage to do a search with multiple keywords

Algorithm's output is plotted in Figure 2, where the y-axis shows the cumulative number of correctly matched step definitions when the length of the possible matches list is limited with a given number (i.e., only x of the most semantically relevant step definitions is considered).

B. *Representation of BDATs with tagged ESGs*

This work utilizes event sequence graphs for modeling BDATs. To model a BDAT as an ESG, ESGs are extended with tags [1].

Definition 4. A tagged ESG is an ESG, where a node or vertex may contain a tag instead of an event.

A tagged ESG is useful in transforming Gherkin scenarios or BDATs to ESGs. Contexts and actions are represented by tags and this way, tags become connection or composition points for ESGs.

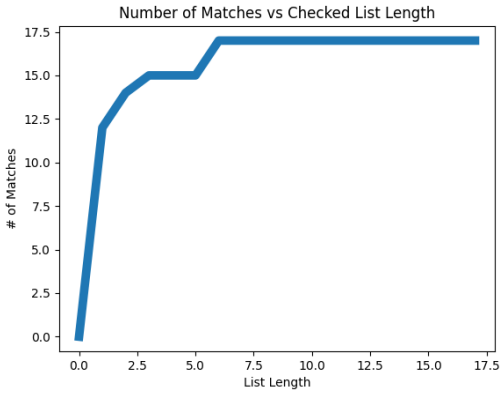

Figure 2. Number of correct matches for a given list length.

For instance, in the following Scenario cart02, *Given* event is tagged with #productPage and *Then* event is tagged with #shoppingBasket. Its ESG representation is shown in Figure 3.

Scenario: cart02 - Adding a product to cart
Given I am on a product detail page #productPage
When I select the amount
And I click the add to cart button
Then the product is added to my shopping cart #shoppingCart

Figure 3. Tagged ESG for Scenario cart02.

Annotating Gherkin clauses with tags and representing BDATs with tagged ESGs enable us to combine BDATs.

C. *Combining two BDATs on tagged ESG*

To combine two BDATs, the following method is proposed. Ending Gherkin clause can be combined with starting Gherkin clause if they have the same tag. This means two Gherkin scenarios can be run in a sequence. We can connect Scenario cart02 with Scenario check01 presented below, where *Given* event is tagged with #shoppingBasket and *Then* event is tagged with #orderConfirmed. ESG representation of Scenario check01 is shown in Figure 4.

Scenario: check01 - Successful checkout
Given I have added an item to my shopping bag #shoppingCart
When I proceed to the check out
And I enter valid delivery details
And I select a payment method
And I confirm the order
Then I am redirected to the thank you page #orderConfirmed

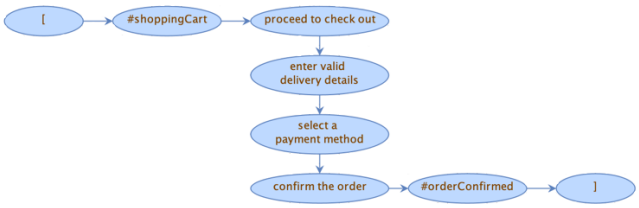

Figure 4. Tagged ESG for Scenario check01.

As seen, tags are used as connection points. Following the method presented in Section III-A, we can combine these two BDATs on a tagged ESG, since both are represented as a tagged ESG. The resulting tagged ESG is shown in Figure 5.

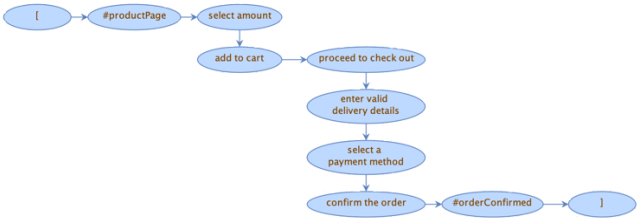

Figure 5. Tagged ESG for combined Scenarios cart02 and check01.

Algorithm 2 outlines the steps explained above.

Algorithm 2. Create Tagged ESG Segments

```
Input: Scenarios
Segments ← { }
For all scenario in scenarios
    entryNode ← ∅
    entryNode.label ← scenario.entryTag
    entryNode.isTag ← true
    prevNode ← entryNode
    For all stepDefinition is scenario.stepDefinitions
        stepNode ← ∅
        stepNode.label ← stepDefinition.text
        stepNode.isTag ← false
        prevNode.next ← stepNode
        prevNode ← stepNode
    exitNode ← ∅
    exitNode.label ← scenario.exitTag
    exitNode.isTag ← true
    prevNode.next ← exitNode
    Segments ← Segments U {entryNode}
Endfor
Output: Segments
```

D. *Finding missing BDATs*

To find missing BDATs, elimination by combination is proposed [1]. As seen above, once two BDATs are combined using a tag, that tag is eliminated. Therefore, first all possible tagged scenarios or their graphical representations, i.e., tagged ESGs, are merged. Algorithm 3 outlines the process of this operation.

Algorithm 3. Merge Tagged ESG Segments

```
Input: Segments
discoveredTags ← { }
For segment in segments
    For node in segment
        if(node.isTag)
            if(node.label in discoveredTags)
                // replace tag node with matched tag node
                // by adding node's descendents to matched node
                discoveredTags[node.label].takeoverNeighbors(node)
            else
                discoveredTags ← discoveredTags U {node}
            Endif
        Endif
    Endfor
Endfor
// remove orphan tags after matching
For segment in segments
    if(segment.length = 1)
        Segments ← Segments / {segment}
    Endif
Endfor
Output: Segments
```

It should be noted that a merged tagged ESG may be merged with another simple or merged tagged ESG. The goal is to reach an ESG without any tags, as shown in Figure 6. After all possible combinations are completed, if a tag remained on a tagged ESG indicates that there is a missing BDAT. If there are more than one tag, that may mean more missing BDATs. The process of tag removal is given in Algorithm 4.

Algorithm 4. Remove Tags from ESG Segments

```
Input: segments
discoveredTags ← { }
For segment in segments
    For node in segment
        if(node.isTag and node.hasAncestor and node.hasDescendant)
            For neighbor in node.neighbors
                node.label ← node.label U neighbor.label
                node.takeoverNeighbors(neighbor)
            Endfor
        Endif
    Endfor
Endfor
Output: segments
```

For instance, in the following Scenario acc03, *Given* event is tagged with #atHome and *Then* event is tagged with #orderDetail.

```
Scenario: acc03 - Check orders
    Given I am logged in on the site #atHome
    When I navigate to my orders
    Then I see a list of my orders
    And I can open an order to see the order details #orderDetail
```

This BDAT is the only Gherkin scenario that has the tag #orderDetail. Since there is no match, it indicates that a BDAT that starts with #orderDetail tag is missing. We can complete this missing BDAT as follows:

```
Scenario: acc10 - Back to order list page
                Given #orderDetail
                When I press OK button
                Then order list page is displayed #orderList
```

As seen in the running example, elimination by combination gives us clues about the connectivity of BDATs. The method proposed here is to check whether all tags are combined. Any tag that is not eliminated suggests a missing BDAT.

E. Composition of BDATs on tagged ESG

After completing the missing BDATs and improving existing BDATs, the BDATs are composed on an ESG. The resulting ESG is shown in Figure 6. Elimination by combination enables us to find five missing BDATs, which are drawn in red on the resulting ESG in Figure 6.

IV. Tool Support

Algorithms 1 to 4 are implemented using Python and are provided at https://github.com/esg4aspl/Connectivity-Improvement-for-Behavior-Driven-Acceptance-Tests along with test data. For NLP operations, Natural Language Processing Toolkit (NLTK) [18] is used. Algorithm 1 is implemented in scenario_matcher.py, where the script takes a list of directories containing Gherkin scenarios and for each directory outputs the per step definition list of step definitions sorted according to semantic similarity. In addition to the output, match rate vs list length is plotted. Algorithms 2 to 4 are implemented in scenario_to_esg.py, where the script takes a directory containing tagged Gherkin scenarios and applies Algorithms 2 to 4 in order, converting scenarios to ESG segments and merging those segments by connecting them at the matching tags.

Once an ESG is ready then CES for edge and for edge-pair coverage can be generated for BDATs. The details of CES generation can be found in [14]. We utilized the TSD tool [19] to generate CES for both coverage criteria. The results are given in Section V-A.

V. Evaluation

For evaluation, the proposed method is applied to an existing test suite for an e-commerce software [20], which is also used as a running example in Section III, and the results are explained in Section V-A. For further evaluation, we asked five teams of graduate students to write BDATs for the same bank ATM software [21] after learning Gherkin and BDATs in a software testing graduate course. Their results are given Section V-B.

A. Evaluation of an E-commerce Software Test Suite

For the existing test suite for an e-commerce software [12], six features out of eight are taken for evaluation. The features locale and newsletter are left. The existing test suite has 15 scenarios, or BDATs, with 64 Gherkin clauses. Clause per scenario ratio is 4.26.

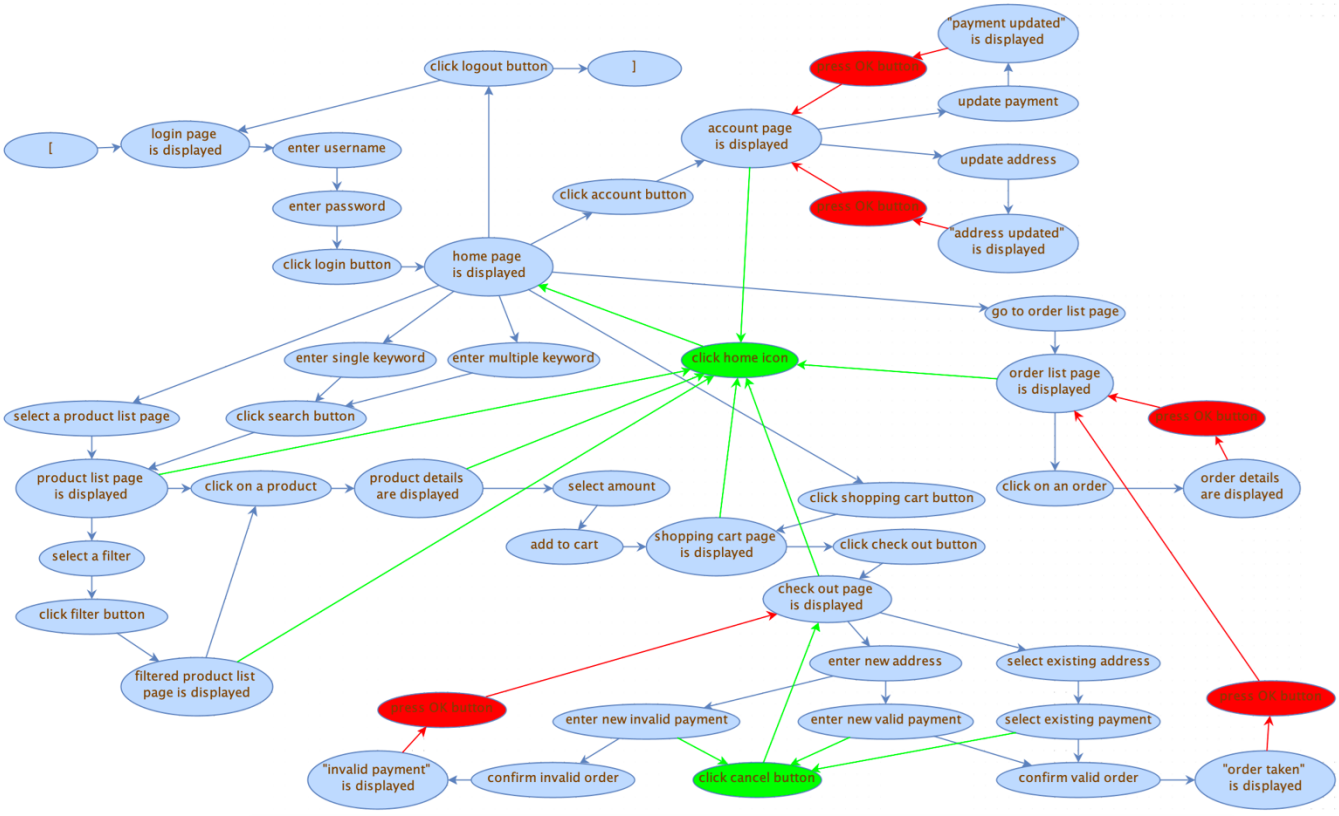

Figure 6. Composed ESG.

After applying the proposed method, we end up with 24 BDATs and 85 Gherkin clauses. There are 9 new scenarios but only 5 of them are missing scenarios. The other 4 scenarios are introduced to simplify and standardize some original scenarios. So, clause per scenario ratio is decreased to 3.54 from 4.26. The comparison of before and after the proposed method is given in Table II. The resulting test suite has the scenarios that are simplified, standardized, and tagged. Moreover, they become composable.

A further analysis of the resulting ESG shows that event sequences are stuck in the child pages of home page. There is no return to home page from child pages, which means that features of the software cannot be tested in sequence. In addition, it is discovered that there is no scenario about cancellation of the check-out process. Those BDATs are added in green to the resulting ESG in Figure 6. It should be noted that the graphical representation of BDATs enables us to perform such an analysis. Without tool support, it is very hard for test designers to conduct such analysis on text represented BDATs.

TABLE II. COMPARISON OF BEFORE AND AFTER PROPOSED APPROACH

Criteria	*Before*	*After*
Number of scenarios	15	24
Number of clauses	64	85
Clause per scenario ratio	4.26	3.54

There is another advantage of the proposed method. Since BDATs are transformed to ESGs and then combined, we have an ESG from which we can automatically generate test sequences, i.e., sequences of BDATs. CES for edge coverage computed by the TSD tool is shown below. There is only one test sequence for the whole BDATs. This shows that the proposed method improves the connectivity in such a way that there is no need for reset operations in test execution.

CES 111 events:
[, login page is displayed, enter username, enter password, click login button, home page is displayed, go to order list page, order list page is displayed, click on an order, order details are displayed, press OK button, order list page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, enter new address, enter new invalid payment, confirm invalid order, "invalid payment" is displayed, press OK button, check out page is displayed, enter new address, enter new invalid payment, click cancel button, check out page is displayed, enter new address, enter new valid payment, click cancel button, check out page is displayed, select existing address, select existing payment, click cancel button, check out page is displayed, enter new address, enter new valid payment, confirm valid order, "order taken" is displayed, press OK button, order list page is displayed, click home icon, home page is displayed, enter multiple keyword, click search button, product list page is displayed, select a filter, click filter button, filtered product list page is displayed, click on a product, product details are displayed, select amount, add to cart, shopping cart page is displayed, click home icon, home page is displayed, enter single keyword, click search button, product list page is displayed, click on a product, product details are displayed, click home icon, home page is displayed, select a product list page, product list page is displayed, click home icon, home page is displayed, click account button, account page is displayed, update payment, "payment updated" is displayed, press OK button, account page is displayed, update address, "address updated" is displayed, press OK button, account page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, select existing address, select existing payment, confirm valid order, "order taken" is displayed, press OK button, order list page is displayed, click home icon, home page is displayed, select a product list page, product list page is displayed, select a filter, click filter button, filtered product list page is displayed, click home icon, home page is displayed, click shopping cart button, shopping cart page is displayed, click check out button, check out page is displayed, click home icon, home page is displayed, click logout button, login page is displayed, enter username, enter password, click login button, home page is displayed, click logout button,],

CES for edge-pair coverage computed by the TSD tool has a complete event sequence of 224 events. The CES for edge-pair coverage is not given here because of space limitations.

B. *Evaluation of an E-commerce Software Test Suite*

Five teams of graduate students wrote BDATs for a bank ATM software [21], which can be found at https://github.com/esg4aspl/Connectivity-Improvement-for-Behavior-Driven-Acceptance-Tests. We applied Algorithm 1 to all five BDAT suites to compare them. The first three list length match percentages (rounded to two digits) are given in Table III. Table III shows that, on the average, 84% of possible tag matches are identified as the first suggestion by Algorithm 1. Average match percentages increase to 87% and 90% respectively, when second and third suggestions are added to the consideration. For TS4, all step definition matches are identified as the first result by the algorithm; while for TS2, all matches are present in top three suggestions.

All the results obtained after applying Algorithm 1 to five BDAT suites are drawn and shown in Figure 7. List length is normalized to account for varying step definition counts. Figure 7 shows that, apart from TS1, all TS have a match rate over 90% within the 5% of the list length (i.e., a correct match is present for 90% of the step definitions within the top 5% of suggested matches list). Match rate further increases to 95% for a 10% list length. For TS1, 90% and 95% match rates are possible at 40% and 65% of the list length respectively.

TABLE III. SCENARIO COUNT, TAG COUNT AND ALGORITHM 1 RESULTS FOR STUDENT GENERATED GHERKIN SCENARIOS

ID	*Scenario count*	*Tagged step definition count*	*Match rate for list length*		
			L=1	*L=2*	*L=3*
BDAT TS1	24	46	57%	61%	63%
BDAT TS2	18	25	88%	92%	100%
BDAT TS3	48	87	87%	92%	94%
BDAT TS4	17	31	100 %	100 %	100%
BDAT TS5	55	109	89%	91%	93%
AVERAGE	32.4	59.6	84%	87%	90%

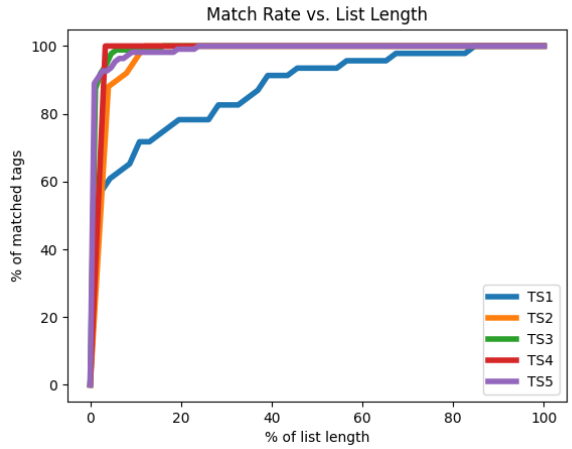

Figure 7. Number of correct matches for five BDAT suites under consideration.

Due to space constraints, we chose ESG of BDAT suite 2 for presentation. Elimination by combination enables us to find one missing BDAT, which is drawn in red on the resulting ESG in Figure 8. Further analysis showed that three scenarios are stuck and cannot lead to exit. For those, scenarios represented in green on the ESG in Figure 8 are added to improve connectivity. Finally, CES for edge coverage computed by the TSD tool is given below. This time there are four CESs. The TSD tool minimizes both the number of CESs and the number of events for efficiency.

CES 3 events: [, insert invalid cash card to ATM, ATM shows error message, eject the card,],

CES 5 events: [, insert valid cash card to ATM, redirect to password page, password page is shown, user clicks take card button, eject the card,],

CES 11 events: [, insert valid cash card to ATM, redirect to password page, password page is shown, enter wrong password, click correction password, clear password, password page is shown, enter wrong password 3 times, freeze the account, display frozen account warning, eject the card,],

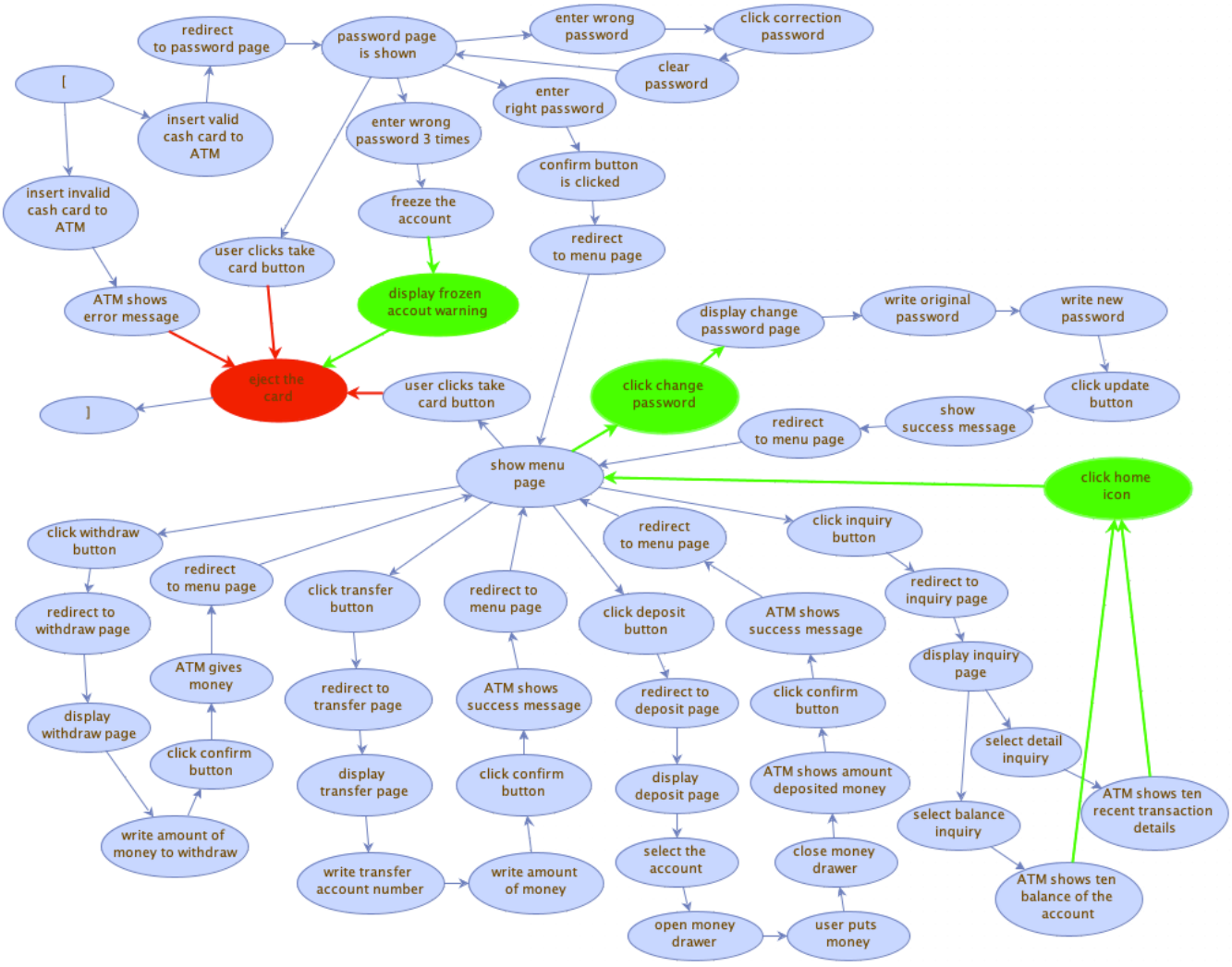

Figure 8. Composed ESG.

CES 60 events: [, insert valid cash card to ATM, redirect to password page, password page is shown, enter right password, confirm button is clicked, redirect to menu page, show menu page, click change password, display change password page, write original password, write new password, click update button, show success message, redirect to menu page, show menu page, click inquiry button, redirect to inquiry page, display inquiry page, select balance inquiry, ATM shows ten balance of the account, click home icon, show menu page, click transfer button, redirect to transfer page, display transfer page, write transfer account number, write amount of money, click confirm button, ATM shows success message, redirect to menu page, show menu page, click deposit button, redirect to deposit page, display deposit page, select the account, open money drawer, user puts money, close money drawer, ATM shows amount deposited money, click confirm button, ATM shows success message, redirect to menu page, show menu page, click withdraw button, redirect to withdraw page, display withdraw page, write amount of money to withdraw, click confirm button, ATM gives money, redirect to menu page, show menu page, click inquiry button, redirect to inquiry page, display inquiry page, select detail inquiry, ATM shows ten recent transaction details, click home icon, show menu page, user clicks take card button, eject the card,],

VI. Discussion

The proposed method assumes that Gherkin clauses can be represented by events. This assumption holds for the selected two BDAT suites used in the evaluation. We were able to represent all possible Gherkin clauses by events.

In the previous section, evaluation of Algorithm 1 results for five BDAT suites showed that Algorithm 1 performs significantly worse for TS1. In fact, excluding TS1 increases the first and third result hit rate in average to 91% and 97% up from 84% and 90%, respectively. To explain this discrepancy, TS1 Gherkin scenarios were examined, and the issue was traced back to the original set of requirements. In the original set of requirements, the system is described as two interacting modules. The team of the TS1 converted these requirements into 2 separate Gherkin features with different perspectives and terminologies. As a result, matching step definitions' semantic similarities were severely weakened across features. 85% of the missed first suggestion matches for TS1 were observed to be between step definitions from different features. This observation also leads us to a known fact that BDATs should be written without considering any design or implementation issues.

The evaluation of the second case reveals that using NLP techniques on written BDATs helps us improve the connectivity of BDATs. Moreover, the proposed method shows that through modeling BDATs, it is possible to generate test sequences automatically. UML use case diagrams and activity diagrams can also be used for modeling BDATs and then automatically generate tests. The research in this area is explained in the related work section.

Scalability of the models is an important concern. ESGs allow us to work on some small and modular models through sub-ESGs [12]-[15] like subroutines. The TSD tool is also designed to support sub-ESGs. This way, it is possible to generate manageable large models. Moreover, these sub-ESGs can be flattened into one large ESG if necessary.

VII. Threats to Validity

One threat to validity is internal validity, which deals with the effects on the evaluation. The selection of BDAT test suite used in evaluation is obtained by searching GitHub repositories. This cannot be considered as random selection. Moreover, the proposed method is applied to the selected BDAT test suite by the author.

Another threat to validity is external validity, which deals with the generalizability of the results. The evaluation in this study is based on a single BDAT test suite. Although this test suite is developed for e-commerce software, which may represent business software generally, evaluation of other BDAT test suites from different domains with the proposed method will help generalize the results.

VIII. Related Work

Tuglular [22] proposed a model-based approach for feature-oriented testing using Event Sequence Graphs (ESGs). In this approach, ESGs are extended to save state and pass it to the following ESG. This way, tests written for features can be combined on state information. However, capturing state is not always possible for acceptance tests.

UML use case diagrams can also be used for modeling BDATs and then automatically generate tests. Gutierrez et al. [23] proposed an approach for working with Gherkin scenarios using UML use case models. They transform from the UML use case diagrams to the Gherkin plain text syntax. They also developed a tool for running Gherkin scenarios in UML as test cases.

Alferez et al. [24] proposed an approach, named AGAC (Automated Generation of Acceptance Criteria), which supports the automated generation of AC specifications in Gherkin. They used UML use case diagrams and activity diagrams to create specifications, derive acceptance criteria from them, and then generate test cases from derived acceptance criteria. UML activity diagrams are not formally defined as directed graphs and therefore, in this work we choose to use formally defined ESGs to benefit from existing algorithms in directed graphs. However, with the help of some theoretical background UML activity diagrams can be used instead of ESGs.

Kudo et al. [25] proposed the software pattern meta model that bridges requirement patterns to groups of scenarios with similar behaviors in the form of test patterns. This meta model is used to describe the behavior of a requirement pattern through a time executable and easy-to-use language aiming at the automatic generation of test patterns.

Wanderley and da Silveria [26] proposed using a mind model specification, which serves as a basis for transforming the definitions of the scenario and generating a conceptual model represented by a UML class diagram. The mind model functions as a bond that represents the business entities, and enables simple association, aggregation, and composition relationships between the entities.

An adjacent area is process discovery in business process management literature. Rozinat and van der Aalst [27] worked on whether event logs conform to the process model and vice versa. They proposed two dimensions of conformance, namely fitness and appropriateness, to be checked along with corresponding metrics. They developed a Conformance Checker within the ProM Framework.

Beschastnikh et al. [28] proposed algorithms for inferring communicating finite state machine models from traces of

concurrent systems, and for proving them correct. They also provided an implementation called CSight, which helps developers find bugs.

Pecchia et al. [29] proposed an approach that employs process mining for detecting failures from application logs. Their approach discovers process models from logs; then it uses conformance checking to detect deviations from the discovered models. They were able to quantify the failure detection capability of conformance checking despite missing events, and its accuracy with respect to process models obtained from noisy logs [29].

As a novel approach, this work aims to transform executable specification in Gherkin language to an ESG. Additionally, this work introduces a novel methodic analysis on BDATs that can reveal missing BDATs.

IX. Conclusion

This paper proposes a method to improve the connectivity of behavior-driven acceptance tests. The method utilizes NLP techniques and ESGs. With the proposed method, the test designer not only finds and completes missing BDATs, but also combines them to know which BDAT can be executed after which BDAT. When the final composition is supplied to the TSD tool, it automatically generates a test sequence that covers all BDATs. So, the proposed method improves the connectivity of BDATs.

As future work, we plan to enhance the developed tool with new capabilities to further aid in the design and application of acceptance tests. Also, as future work, our goal is to enhance the tool with ontologies so semantically related scenarios are easily decoded. Moreover, we plan to use UML activity diagrams instead of event sequence graphs and compare their advantages and disadvantages. Finally, we will apply all these improvements to large Gherkin-based specifications and acceptance criteria.

References

[1] T. Tuglular, “On the Composability of Behavior Driven Acceptance Tests,” in The Seventh International Conference on Advances and Trends in Software Engineering (SOFTENG 2021), 2021, pp. 1–4.

[2] M. G. Cavalcante and J. I. Sales, "The Behavior Driven Development Applied to the Software Quality Test," Proc. 14th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, 2019, pp. 1–4.

[3] Cucumber Gherkin. https://cucumber.io/docs/gherkin/reference/. [retrieved: March, 2021].

[4] Gauge. https://gauge.org. [retrieved: March, 2021].

[5] T. Tuglular and S. Şensülün, “SPL-AT Gherkin: A Gherkin Extension for Feature Oriented Testing of Software Product Lines,” in IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2019, vol. 2, pp. 344–349.

[6] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” Information processing & management, vol. 24, no. 5, pp. 513–523, 1988.

[7] P. Sitikhu, K. Pahi, P. Thapa, and S. Shakya, “A comparison of semantic similarity methods for maximum human interpretability,” in 2019 artificial intelligence for transforming business and society (AITB), 2019, vol. 1, pp. 1–4.

[8] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan, “Cosine similarity to determine similarity measure: Study case in online essay assessment,” in 2016 4th International Conference on Cyber and IT Service Management, 2016, pp. 1–6.

[9] S. Kannan et al., “Preprocessing techniques for text mining,” International Journal of Computer Science & Communication Networks, vol. 5, no. 1, pp. 7–16, 2014.

[10] S. Vijayarani, M. J. Ilamathi, and M. Nithya, “Preprocessing techniques for text mining-an overview,” International Journal of Computer Science & Communication Networks, vol. 5, no. 1, pp. 7–16, 2015.

[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), 2016, pp. 183–215.

[12] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," Applied Intelligence, 26(2), 2007, pp. 161–174.

[13] F. Belli, C. J. Budnik, and L. White, "Event based modelling, analysis and testing of user interactions: approach and case study," Software Testing, Verification and Reliability, 16(1), 2006, pp. 3–32.

[14] F. Belli and C. J. Budnik, "Minimal spanning set for coverage testing of interactive systems," International Colloquium on Theoretical Aspects of Computing. Springer, Berlin, Heidelberg, 2004, pp. 220–234.

[15] T. Tuglular, C. A. Muftuoglu, F. Belli, and M. Linschulte, "Event-based input validation using design-by-contract patterns," Proc. 20th International Symposium on Software Reliability Engineering, ISSRE’09, IEEE Press, 2009, pp. 195–204.

[16] G. Chartrand, “Introductory Graph Theory,” Courirer Corporation, 1977.

[17] E. Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software," Addison-Wesley Professional, 2003.

[18] S. Bird, E. Klein, and E. Loper, Natural language processing with Python, 1st ed. Beijing ; Cambridge [Mass.]: O’Reilly, 2009.

[19] TestSuiteDesigner. http://download.ivknet.de/. [retrieved: March, 2021].

[20] Barzilay, "Example of an ECommerce cucumber web test automation suite,". https://github.com/spriteCloud/ecommerce-cucumber-web-test-automation-suite. [retrieved: March, 2021].

[21] Team-111, “Bank ATM software,” https://github.com/Team-1111/bankbankatm. [retrieved: June, 2021]

[22] T. Tuglular, "Event sequence graph-based feature-oriented testing: A preliminary study," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, 2018, pp. 580–584.

[23] J. J. Gutiérrez, I. Ramos, M. Mejías, C. Arévalo, J. M. Sánchez-Begines, and D. Lizcano, "Modelling Gherkin Scenarios Using UML," Proc. 26th International Conference on Information Systems Development (ISD), 2017.

[24] M. Alferez, F. Pastore, M. Sabetzadeh, L. Briand, and J. R. Riccardi, "Bridging the gap between requirements modeling and behavior-driven development," 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2019, pp. 239–249.

[25] T. N. Kudo, R. F. Bulcão-Neto, and A. M. Vincenzi, "A conceptual metamodel to bridging requirement patterns to test patterns," Proc. of the XXXIII Brazilian Symposium on Software Engineering. 2019, pp. 155–160.

[26] F. Wanderley and D. S. da Silveria, "A framework to diminish the gap between the business specialist and the software designer," 2012 Eighth International Conference on the Quality of Information and Communications Technology. IEEE, 2012, pp. 199–204.

[27] A. Rozinat and W.M.P. van der Aalst, "Conformance testing: Measuring the fit and appropriateness of event logs and process models," Proc. 4th Business Process Management Workshops, Springer, 2006, pp. 163–176.

[28] I. Beschastnikh, Y. Brun, M.D. Ernst, and A. Krishnamurthy, "Inferring models of concurrent systems from logs of their behavior with CSight," Proc. 36th International Conference on Software Engineering, ACM, 2014, pp. 468–479.

[29] A. Pecchia, I. Weber, M. Cinque, and Y. Ma, “Discovering process models for the analysis of application failures under uncertainty of event logs,” Knowledge-Based Systems, vol. 189, p. 105054, 2020.