2018 IEEE International Conference on Software Quality, Reliability and Security Companion

Event Sequence Graph-based Feature-oriented Testing: A Preliminary Study

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

Abstract— **This paper proposes a model-based approach for feature-oriented testing using event sequence graphs (ESGs). ESGs are used to generate test cases automatically for positive and negative testing. To fit ESG models to feature-oriented testing, two new improvements on ESGs are proposed. The first improvement is on repetitive use of refinement ESG and the second improvement is saving state in an ESG and passing it to the following ESG. This is a work towards communicating hierarchical ESGs. The preliminary results demonstrate the feasibility of the proposed approach. The proposed approach improves testability of features.**

Keywords-model-based testing; event sequence graphs; feature-oriented testing

I. INTRODUCTION *(HEADING 1)*

In Scrum, a feature is a collection of user stories. Although a feature can actually be considered as an epic, epic is an umbrella for features. It can be said that an epic contains a number of features and user stories, whereas a feature contains only a number of user stories. The hierarchical nature of ESGs fit very well with this approach of Scrum. The suggestion in this paper is to model epics, features, and user stories with ESGs and call respective ESGs as epic ESG, feature ESG, and user story ESG. An example for all these is given in Section 3.

The paper proposes two new improvements on ESGs to support feature-oriented testing. The first improvement is on self-repeating node in ESG that is refined by another ESG and the second improvement is saving state in an ESG and passing it to the proceeding ESG. Both requirements and their exploration in ESGs are shown with a real-life running example in Section 3. The running example is simplified to show the proposed approach and its feasibility.

The paper is organized as follows: In the next section, the formal definitions of ESGs are given along with examples and figures. In the third section, the proposed approach is explained with an example. The fourth and last section concludes the paper.

II. THEORETICAL BACKGROUND

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 testing graphical user interfaces. These requirements are satisfied by event sequence graphs [1].

Differing from the notion of finite-state automata (FSA), inputs and states are merged in ESG, hence they are turned into "events" to facilitate ease of 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 [1]. Definitions from 1 to 4 and related examples and figures are taken exactly as they are from [2,3,4,5].

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 $v \neq \xi, \gamma$.

Ξ (*ESG*), Γ (*ESG*) represent the entry nodes and exit nodes of a given ESG, respectively. 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 is 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 of them 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 Fig. 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

978-1-5386-7839-8/18/$31.00 ©2018 IEEE
DOI 10.1109/QRS-C.2018.00102

predecessors of *v*. Note that $N^-(v)$ is empty for an entry $\xi \in \Xi$ and $N^+(v)$ is empty for an exit $\gamma \in \Gamma$.

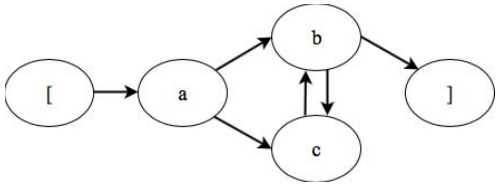

Figure 1. An ESG with a as entry and b as exit and pseudo vertices [,].

Definition 2. Let *V*, *E* be defined as in Definition 2. 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. In particular, 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 invoke no interim system responses during user-system interaction, i.e., it may consist of consecutive user inputs and 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 → ⋯ →(final) system response.

Note that a CES may invoke no interim system responses during user-system interaction, i.e., it may consist of consecutive user inputs and a final system response. To keep the size of ESGs tractable, the ESGs topmost layer can be refined in several modularization steps resulting in a hierarchical set of ESGs. In Fig. 2, an example of a vertex v being refined by another ESG is given. The figure also contains the completed version without refinement.

Definition 4. Given an ESG, say $ESG_1 = (V_1, E_1, \Xi_1, \Gamma_1)$, a vertex $v \in V_1$, and an ESG, say $ESG_2 = (V_2, E_2, \Xi_2, \Gamma_2)$. Then replacing *v* by ESG_2 produces a *refinement* of ESG_1, say $ESG_3 = (V_3, E_3, \Xi_3, \Gamma_3)$ with $V_3 = V_1 \cup V_2 \setminus \{v\}$, and $E_3 = E_1 \cup E_2 \cup E_{pre} \cup E_{post} \setminus E_{1\ replaced}$ ('\': set difference operation), wherein $E_{pre} = N^-(v) \times \Xi_2$ (connections of the predecessors of *v* with the entry nodes of ESG_2), $E_{post} = \Gamma_2 \times N^+(v)$ (connections of exit nodes of ESG_2 with the successors of *v*), and $E_{1\ replaced} = \{(v_i, v), (v, v_k)\}$ with $v_i \in N^-(v)$ and $v_k \in N^+(v)$ (replaced arcs of ESG_1).

III. Proposed Approach

The running example is an epic of online shopping. It is composed of two user stories, namely login and paying shopping cart and a feature, namely doing shopping. As seen in Fig. 3, "Do Shopping" node is refined with "Search and Select Product" and "Add to Shopping Cart", which may be considered as user stories, since they can be further refined as seen in Fig. 4 and can be tested alone. It can be said that if a node in an ESG is refined by two levels, then it is a feature. If a node in an ESG is refined only by one level, then it is a user story. So, top level of Fig. 3 is an epic ESG for online shopping epic, the "Do Shopping" node represents a feature ESG, and the "Search and Select Product" node represents a user story ESG. As seen in the example, ESG is a good fit in modeling epics, features, and user stories in Scrum.

In Fig. 4, "Search and Select Product" and "Add to Shopping Cart" are refined by two ESGs. Modelling "Do Shopping" feature as in Fig. 4 improves testability of the epic, since controllability and observability of each leaf ESG representing, i.e. refining, user stories of the feature is very high.

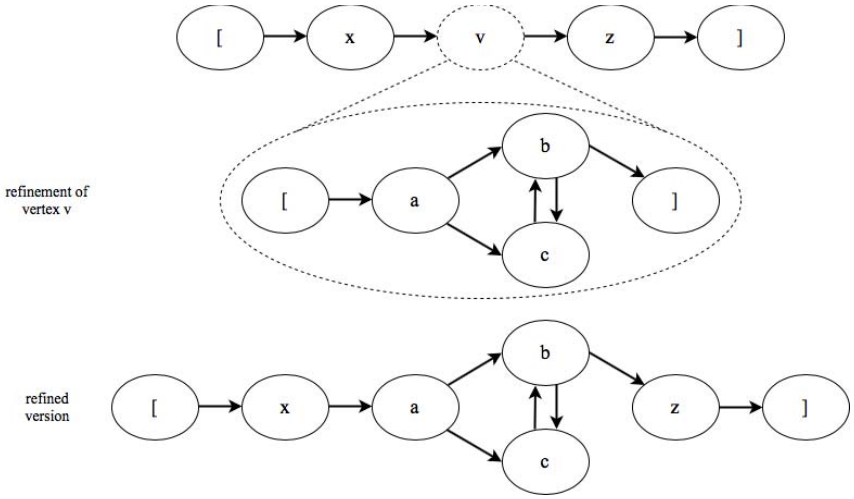

Figure 2. Refinement of a vertex v and its embedding in the refined ESG.

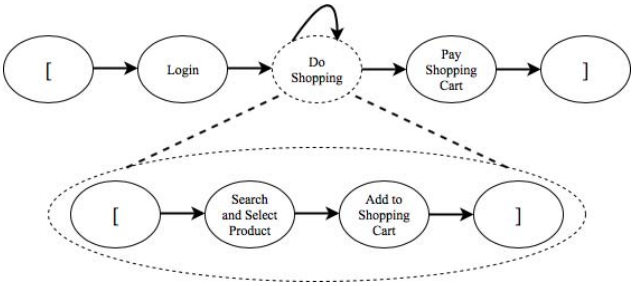

Figure 3. ESG for online shopping epic.

Modeling "Do Shopping" feature as in Fig. 4 introduces two new cases for ESGs. The first one is having self-repeating node in ESG that is refined by another ESG. The second is saving state in one ESG and passing it to the next ESG in execution order. Exploring these two new cases is the novelty and contribution of this paper in addition to fitting ESGs to epics, features, and user stories in Scrum.

Having a self-repeating vertex in ESG that is refined by another ESG is a new property for ESGs. When explored on ESG given in Fig. 2 by making the vertex v self-repeating, the ESG given in Fig. 5 is obtained. The mechanism of refining self-repetition is proposed in Definition 5.

Definition 5. Given an ESG, say $ESG_1 = (V_1, E_1, \Xi_1, \Gamma_1)$, a vertex $v \in V_1$, and an ESG, say $ESG_2 = (V_2, E_2, \Xi_2, \Gamma_2)$. Then replacing v by ESG_2 produces a *refinement* of ESG_1, say $ESG_3 = (V_3, E_3, \Xi_3, \Gamma_3)$ with $V_3 = V_1 \cup V_2 \setminus \{v\}$, and $E_3 = E_1 \cup E_2 \cup E_{pre} \cup E_{post} \setminus E_{1\ replaced}$ ('\': set difference operation) as in Definition 4. If v is a self-repeating vertex and to be refined, then all levels are refined till up to self-repeating vertex v and then an edge from exit node of final refinement to entry node of final refinement is added. In case, there are multiple exit nodes and/or multiple entry nodes of final refinement, then each exit node of final refinement is connected to every entry node of final refinement.

When the new Definition 5 is applied to the running example given in Fig. 4, stepwise refinement is shown Fig. 6 and Fig. 7. At each step, one level refinement is performed. Since there are two levels of refinement for "Do Shopping" vertex, refinement is performed in two steps given in Fig. 6 and Fig. 7. After refinement is completed, as proposed in Definition 5, an edge from each exit node of final refinement to entry node of final refinement is added, which are edges from "Add to Shopping Cart" to "Enter Product Name" and from "Search" to "Enter Product Name".

Saving state in one ESG and passing it to the next ESG in execution order is a new property for ESGs. In the running example, the shaded rectangle vertices seen in Fig. 4 represent the ESG state, which is passed to the next ESG. The mechanism of passing state from one ESG to another is proposed in Definition 6.

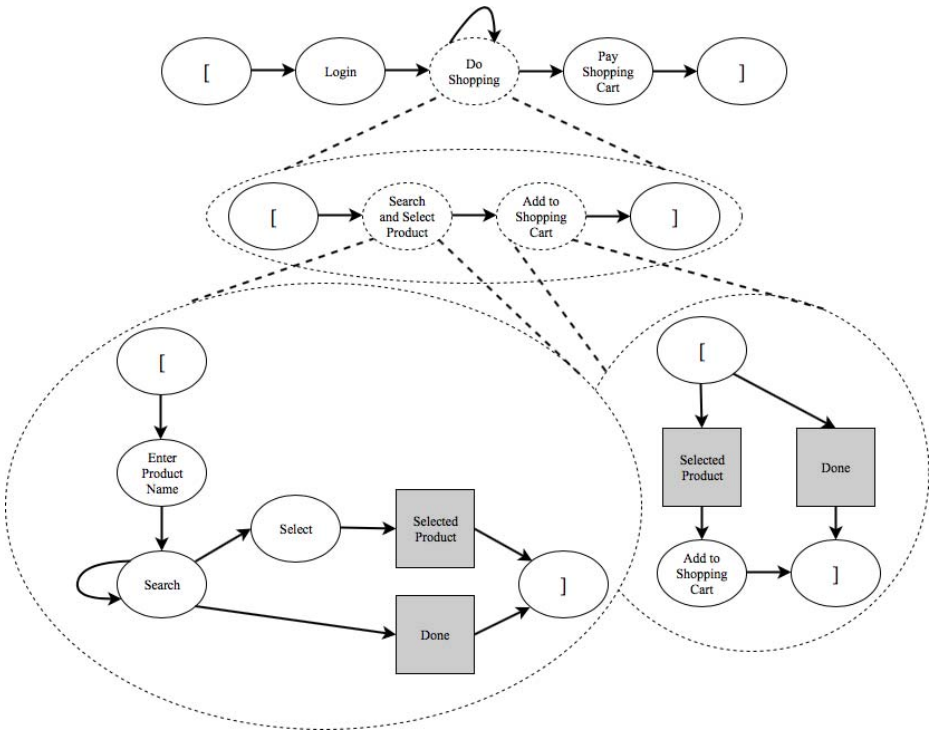

Figure 4. Refinement of ESG for "Do Shopping" feature.

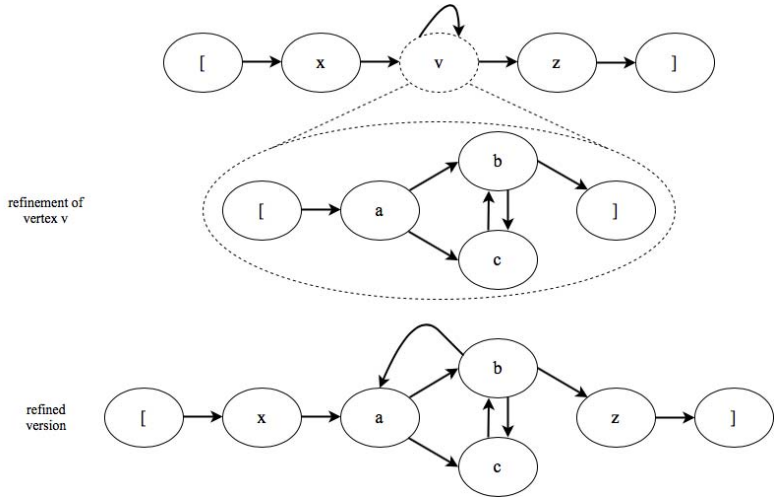

Figure 5. Refinement of a self-repeating vertex v and its embedding in the refined ESG.

The state information vertex or vertices will become exit node(s) for an ESG if it wants to pass state to the next ESG. The next ESG must have the same state information vertex or vertices as entry node(s), so that a mapping can be achieved and both ESGs can be resolved into a single ESG by removing state information vertices. The ability to pass state information from one ESG to another enables ESG-based models to be smaller and more flexible, so that they can easily represent user stories.

Definition 6. Given an ESG, say $ESG_1 = (V_1, E_1, \Xi_1, \Gamma_1)$, a vertex $v \in V_1$, and an ESG, say $ESG_2 = (V_2, E_2, \Xi_2, \Gamma_2)$. Having Γ_1 as state node(s) matching exactly with Ξ_2 as state nodes then enables ESG_1 to pass state information to ESG_2. So, they become communicating ESGs.

Moreover, ESG_1 can be combined with ESG_2 with gluing Γ_1 and Ξ_2 to have a combined ESG say $ESG_3 = (V_3, E_3, \Xi_3, \Gamma_3)$ through eliminating Γ_1 and Ξ_2 and having Ξ_1 and Γ_2 become Ξ_3, Γ_3, respectively.

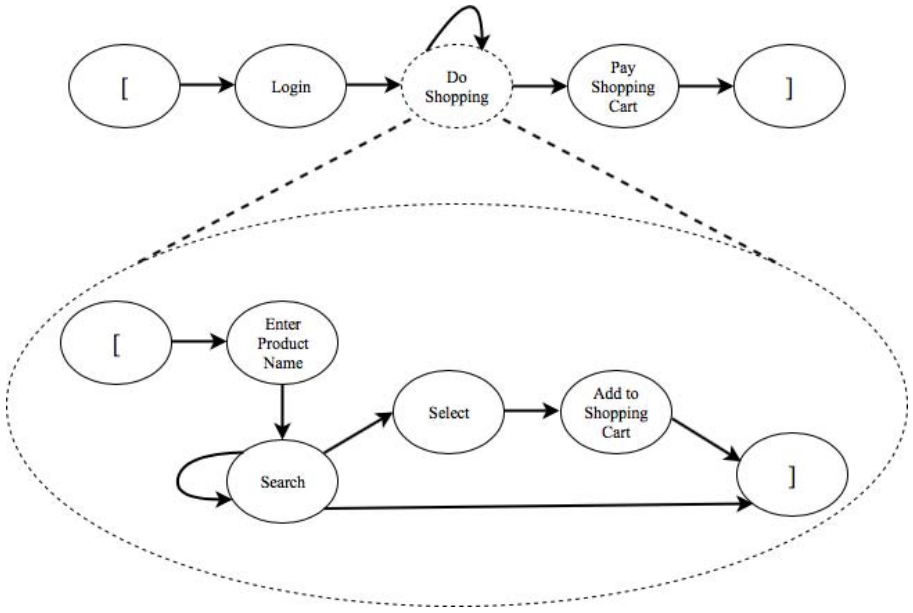

Figure 6. Refinement of a self-repeating vertex v and its embedding in the refined ESG (first step).

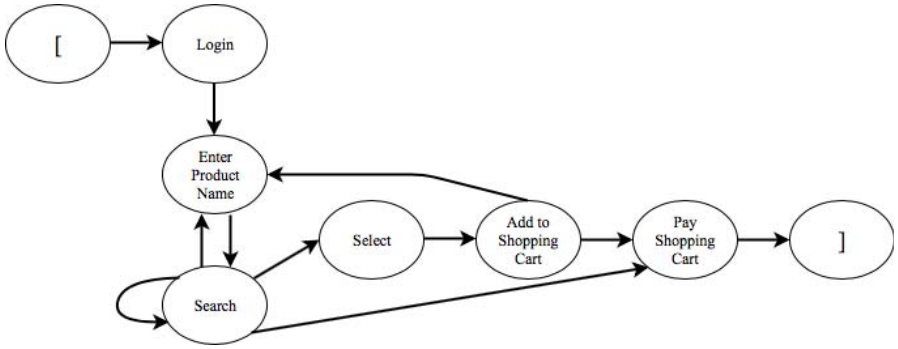

Figure 7. Refinement of a self-repeating vertex v and its embedding in the refined ESG (second step).

The proposed approach improves testability of epics, features, and user stories. User stories are expected to deliver business value and should be able to work stand alone. Hence, they ought to be tested stand alone and later when combined with other user stories, they ought to be tested together. With communicating hierarchical ESGs enabled to model feature-oriented software, ESGs can be used for feature-oriented testing.

IV. CONCLUSION

The paper first discusses modeling epics, features, and user stories in ESG. With an example, it is shown that ESGs can be a good fit in modeling epics, features, and user stories in Scrum. Then the paper proposes two new improvements on ESGs to support feature-oriented testing. The first improvement is on self-repeating node in ESG that is refined by another ESG and the second improvement is saving state in an ESG and passing it to the proceeding ESG. This is a work towards communicating hierarchical ESGs. The preliminary results demonstrate the feasibility of the proposed approach. Moreover, the proposed approach improves testability of features in Scrum. As a future work, it is planned to integrate ESGs with Scrum. In addition to that, the applicability of passing ESG state to another ESG will be further investigated.

REFERENCES

[1] 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, 183-215.

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

[3] 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.

[4] B. Kruger and M. Linschulte, "Cost Reduction through Combining Test Sequences with Input Data," Proc. Sixth International Conference on Software Security and Reliability Companion (SERE-C), IEEE Press, 2012, pp. 207–216.

[5] 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.