Tutorial at SOFTENG 2021

Model-Based Event Sequence Testing of Graphical User Interfaces

Tugkan Tuglular

Izmir Institute of Technology, Turkey

tugkantuglular@iyte.edu.tr

Assoc. Prof. Tugkan TUGLULAR, Ph.D.

Izmir Institute of Technology, Turkey

April 2021

Tugkan Tuglular received the B.S., M.S., and Ph.D. degrees in Computer Engineering from Ege University, Turkey, in 1993, 1995, and 1999.

He worked as a research associate at Purdue University from 1996 to 1998.

He has been with Izmir Institute of Technology since 2000.

After becoming an Assistant Professor at Izmir Institute of Technology, he worked as Chief Information Officer in the university from 2003-2007.

In addition to his academic duties, he acted as IT advisor to the Rector between 2010-2014.

In 2018, he became an Associate Professor in the Department of Computer Engineering of the same university.

He has more than 70 publications and an active record of duties with international and national conferences.

His current research interests include model-based testing and software quality with machine learning support.

Outline

- Motivation
  - Invalid inputs at interfaces, especially in graphical user interfaces
- Model-based Testing
  - Event Sequence Graphs
  - Test Suite Designer Tool
- Applications
  - Input Contract Testing

Introduction

- “Data is the lifeblood of software; when it is corrupt, the software is as good as dead.” (Whittaker, 2001)
- Experiences have shown that inputs should be validated thoroughly to prevent denial of service, intrusion and system crashes.

Bad Experiences

- Mars Climate Orbiter
  - Smashed into the planet
  - Software failure to convert English measures to metric measures
  - Loss of US$ 165 M
- Ariane 5
  - Destroyed by its automated self-destruct system
  - Data conversion from a 64-bit floating point to 16-bit signed integer value caused an arithmetic overflow
  - Loss of US$ 370 M

Motivation

- Software developers and testers must consider every single input from every external resource.
- Deciding which inputs to trust and which to validate is a constant balancing act for software and its developers.
- Preventing invalid input from ever getting to the application in the first place is possible only at interfaces such as GUIs.

Graphical User Interfaces

- Graphical User Interfaces (GUIs)
  - add up to half or more of the source code
- Invalid inputs
  - supplied values may violate design-by-contract conditions
- Invalid sequence of actions
  - order of actions may cause unexpected operations and outputs

What are Models?

- Models are abstractions used to represent and communicate what is important, devoid of unnecessary detail, and to help developers deal with the complexity of the problem being investigated or the solution being developed.

https://www.open.edu/openlearn/science-maths-technology/introduction-software-development/content-section-6

Advantages of Model-based Testing

- It starts with specifications by reinforcing the idea that Quality Assurance involvement belongs at the beginning of the analysis phase.
- It forces testability into the product design when talking about the creation of models for a new/modified feature.
- It typically finds specification and design bugs even before the code exists.
- The automatic test suite generation will increase testing thoroughness, test coverage is guaranteed, and test suite maintenance is minimized.

https://saucelabs.com/blog/the-challenges-and-benefits-of-model-based-testing

Model-based Testing Process

- Start by modeling the Software Under Test (SUT)
- Derive test cases from the model
- Execute test cases
  - use model as test oracle
  - record coverage
  - trace to model
- Modify model as needed
- Repeat steps

P. C. Jorgensen, Software Testing: A Craftsman's Approach, 4th Edition, Auerbach Publications, 2013.

Modeling Graphical User Interfaces

- Graphical user interfaces (GUIs)
  - simplify down to entering values & clicking buttons
  - how to model both user actions?
  - what about software actions?
  - change perspective and look from the receiver of these actions
  - they are all impulses or events
- Event-based modeling of GUIs

Event-based Modeling

- Event-based models introduced
  - Event Sequence Graphs (Belli, 2001)
  - Event Flow Graphs (Memon et al., 2001)
- Nodes are interpreted in both models as events of an event set.
- They can be used for GUI modeling and test generation.

Advantages of Event-based Approach

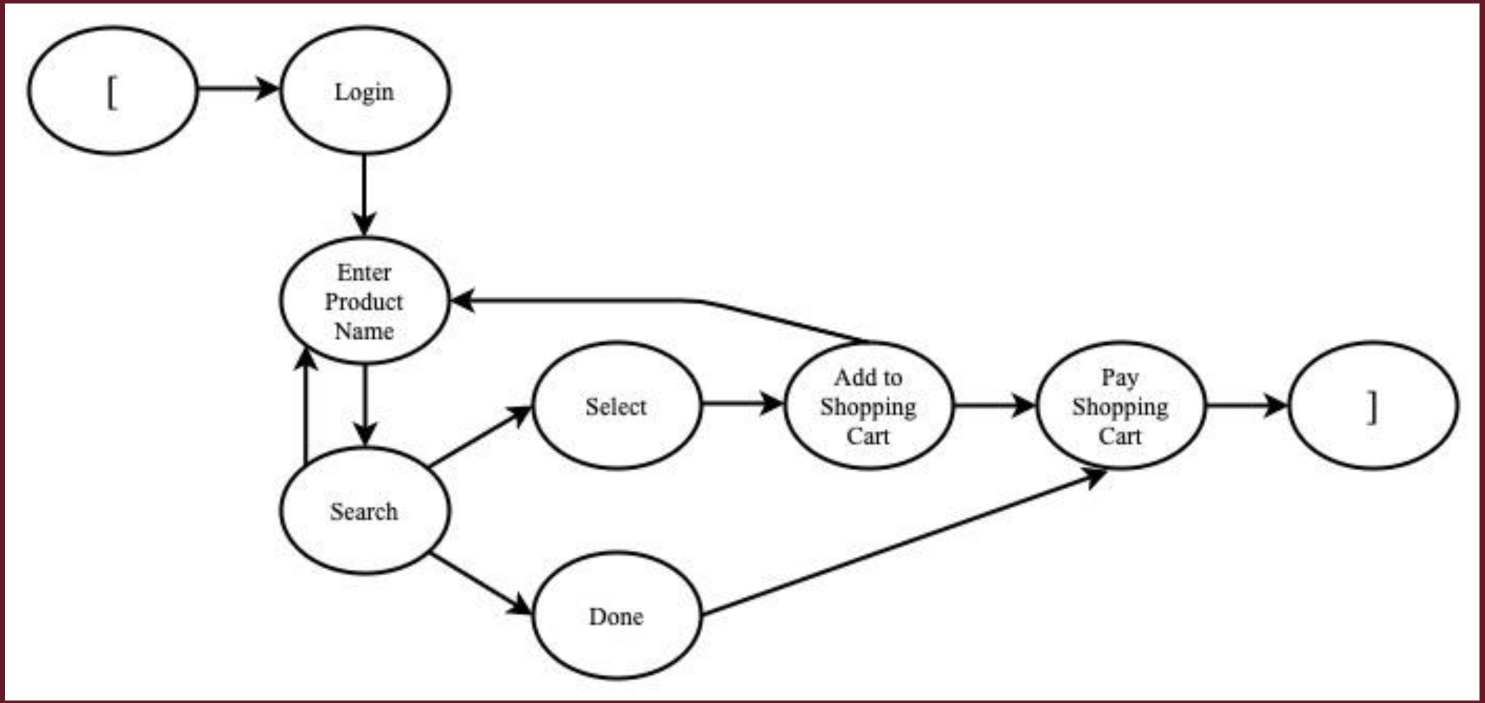
- Testability is dominated by two practical problems:
  - How to provide the test values to the software
  - How to observe the results of test execution
- Controllability
  - How easy it is to provide a program with the needed inputs, in terms of values, operations, and behaviors
- Observability
  - How easy it is to observe the behavior of a program in terms of its outputs, effects on the environment and other hw and sw components

P. Ammann and J. Offutt, Introduction to Software Testing, 2nd Edition, Cambridge University Press, 2016.

Event Sequence Graphs (ESGs)

- Event-based formal model
  - inputs and actions are merged as events and assigned to the vertices of an event transition graph.

A Shopping Cart Application



Event Sequence Graphs (ESGs)

An *event sequence graph* ***ESG*** $= (V, E, \Xi, \Gamma)$ is a directed graph where

$V \neq \emptyset$ is a finite set of vertices (nodes),

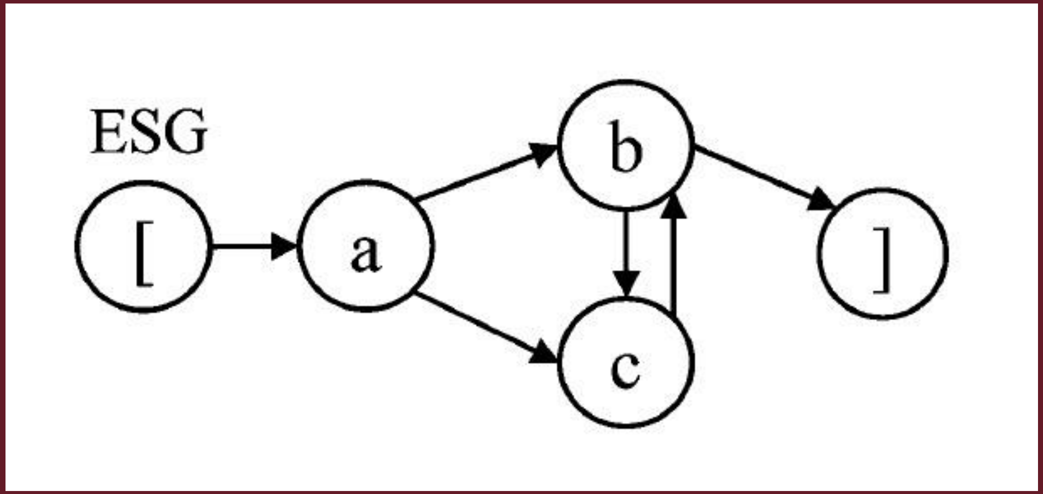
$E \subseteq V \times V$ is a finite set of edges (arcs),

$\Xi, \Gamma \subseteq V$ are finite sets of distinguished vertices

with $\xi \in \Xi$, and $\gamma \in \Gamma$, called entry nodes and exit nodes.

Event Sequence Graphs (ESGs)

- An ESG with a as entry and b as exit and pseudo vertices [,]
- Each edge marked as a legal Event Pair (EP).

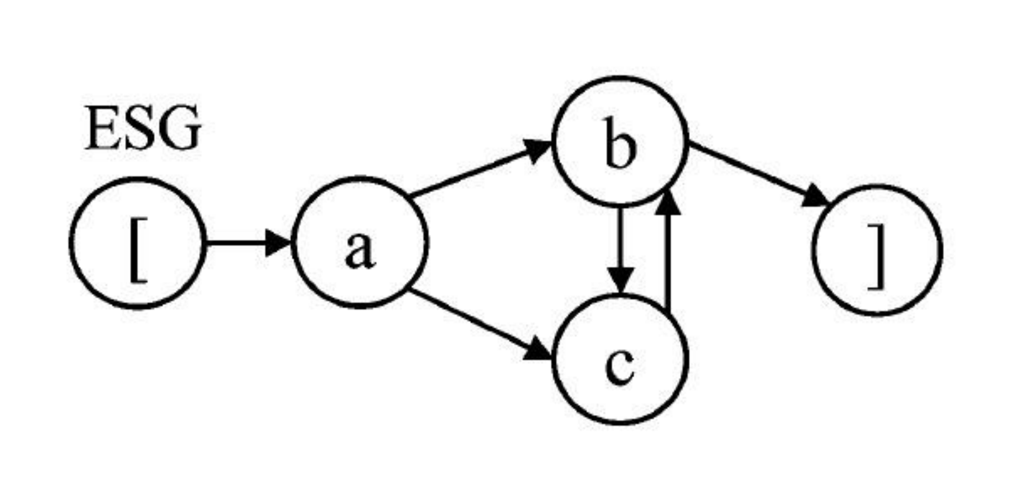


EPs:

a, b
a, c
b, c
c, b

Event Sequence Graphs (ESGs)

- Complete event sequence (CES) represents a walk through the ESG.

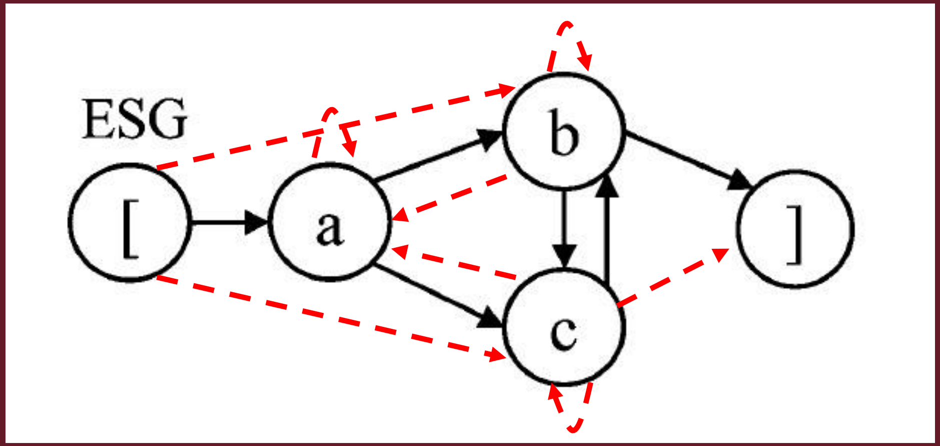


CES:

3: [, a, c, b,],
4: [, a, b, c, b,],

Event Sequence Graphs (ESGs)

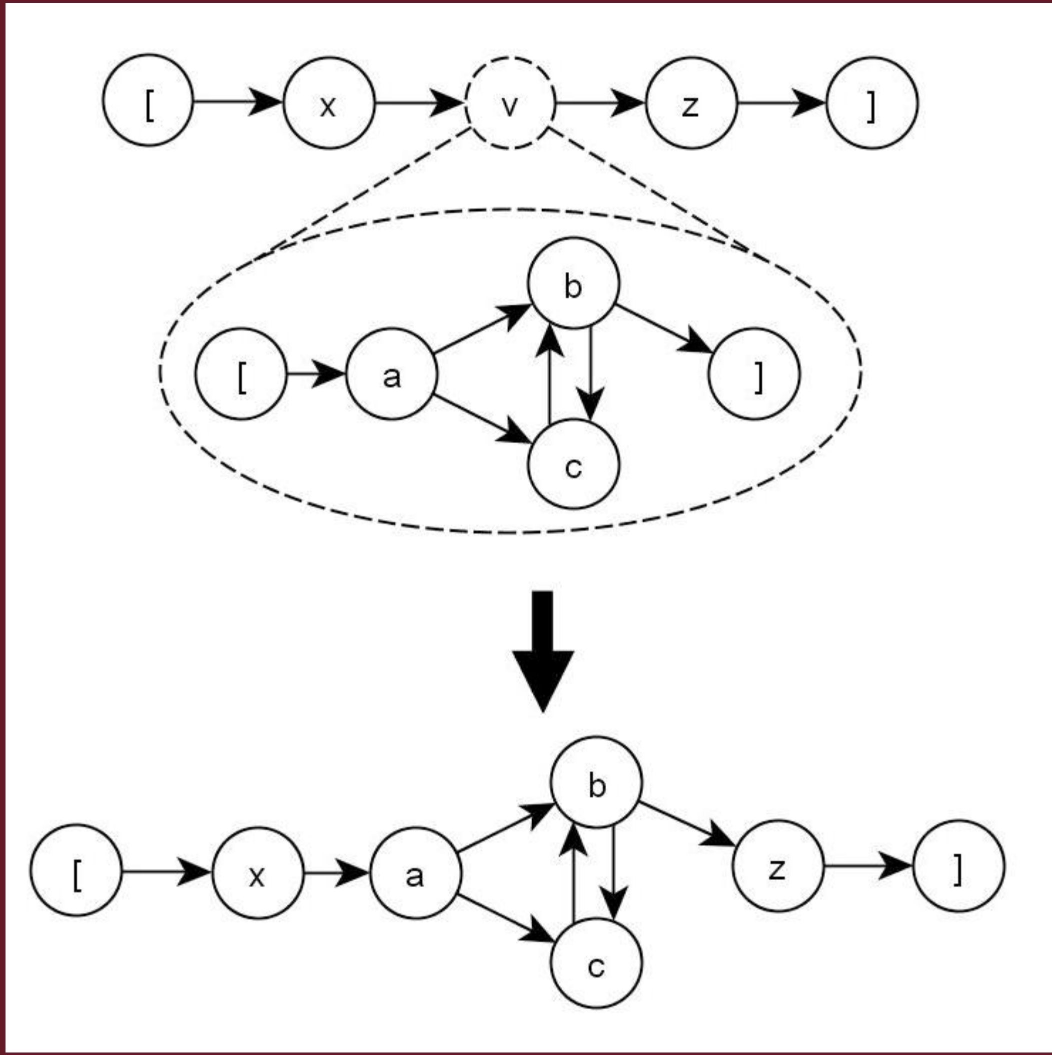
- Faulty (or illegal) Event Pairs (FEPs) are introduced as the edges of the corresponding ESG (red edges)
  - Faulty CESs (FCESs) are constructed using FEPs.



FCES:

2: [, a, a,
3: [, a, b, a,
3: [, a, b, b,
3: [, a, c, a,
3: [, a, c, c,
1: [, b,
1: [, c,

Event Sequence Graphs (ESGs)



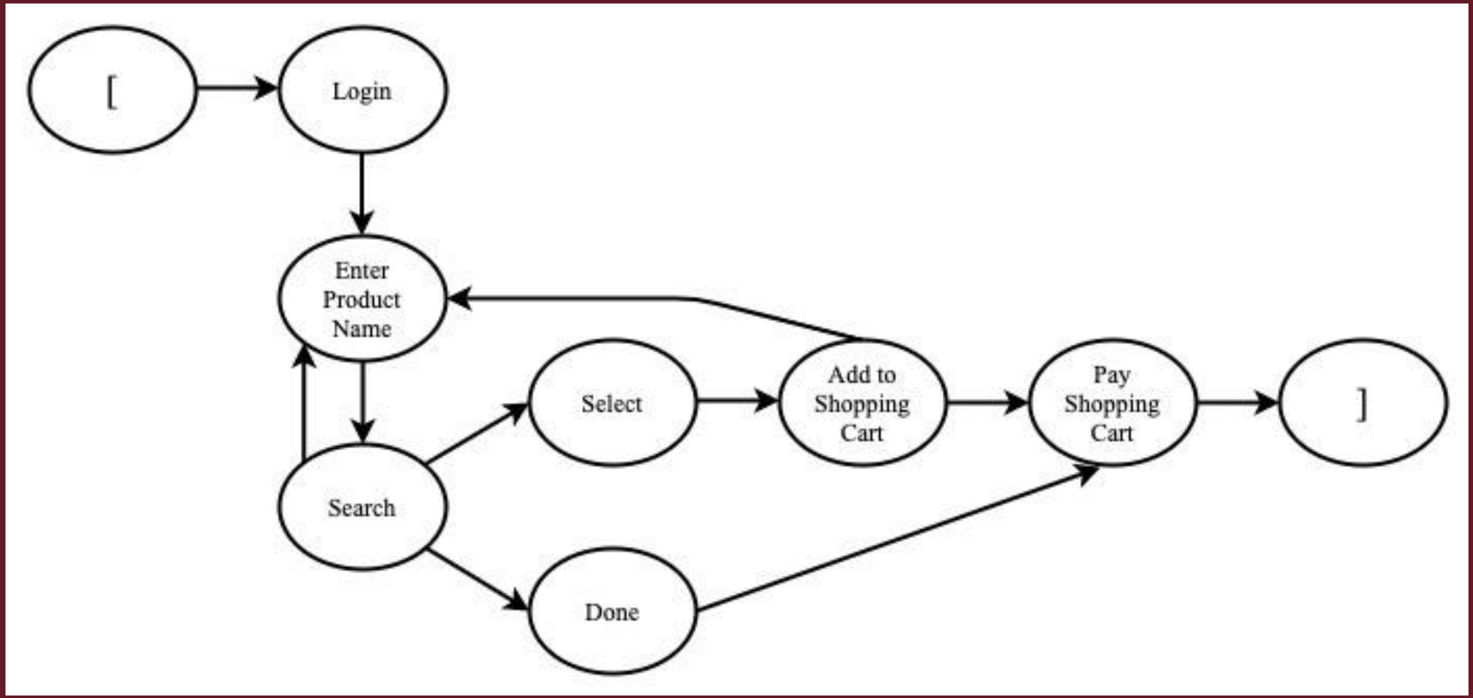
- Refinement of a vertex v and its embedding in the refined ESG

CES:

6: [, x, a, b, c, b, z,],
5: [, x, a, c, b, z,],

Event Sequence Graphs (ESGs)

A Shopping Cart Application



CES:

13: [, Login, Enter Product Name, Search, Search, Enter Product Name, Search, Select, Add to Shopping Cart, Enter Product Name, Search, Select, Add to Shopping Cart, Pay Shopping Cart,],

4: [, Login, Enter Product Name, Search, Pay Shopping Cart,],

ESG Test Generation

- ***Input:*** ESG
- ***Output:*** Test set with respect to model-based coverage criterion
- Two objectives for the test case generation procedure:
  - generation of CESs,
  - generation FCESs from the complement of ESG.
- Test case generation algorithm generates tests that cover both;
  - All event pairs in ESG,
  - All faulty event pairs of the CESG.

ESG Test Generation Algorithm

```
Input: an ESG with
 n:= number of the functional units (modules) of the system that fulfill a well-
      defined task
 length:= required length of the event sequences to be covered
    FOR unit 1 TO n DO
    BEGIN
       Generate appropriate ESG and ESG‾
       FOR k:=2 TO length DO
       BEGIN
          Cover all ESs of length k by means of CESs subject to
          minimizing the number and total length of the CESs
       END
       Cover all FEPs of unit by means of FCESs subject to
       minimizing the total length of the FCESs
    END
    Apply the test set given by the selected CESs and FCESs to the SUT.
    Observe the system output to determine whether the system response is in
    compliance with the expectation.
```

k = 2 means

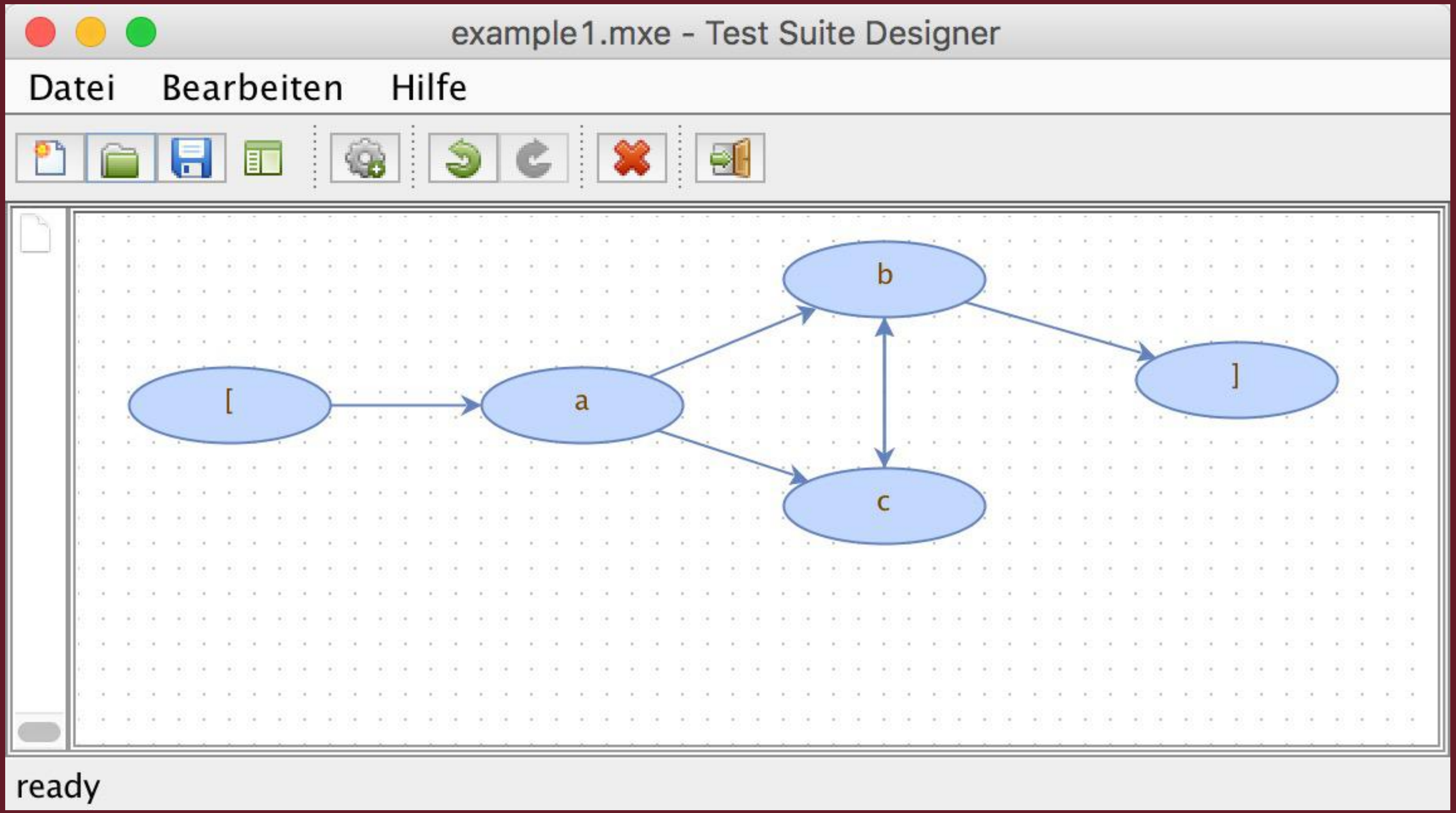
edge (EP) coverage

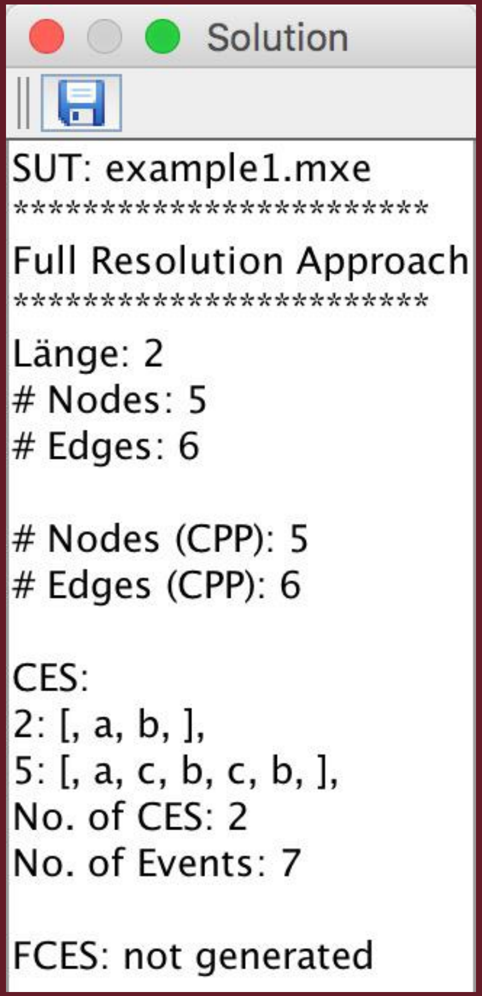
k = 3 means

Edge pair (ET) coverage

…

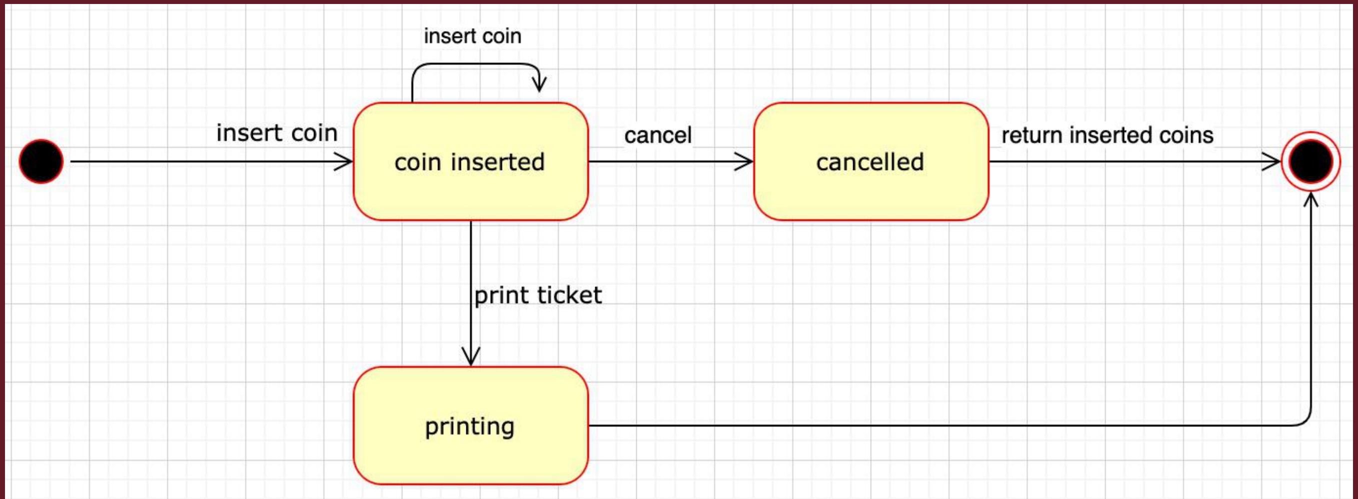
ESG Tool

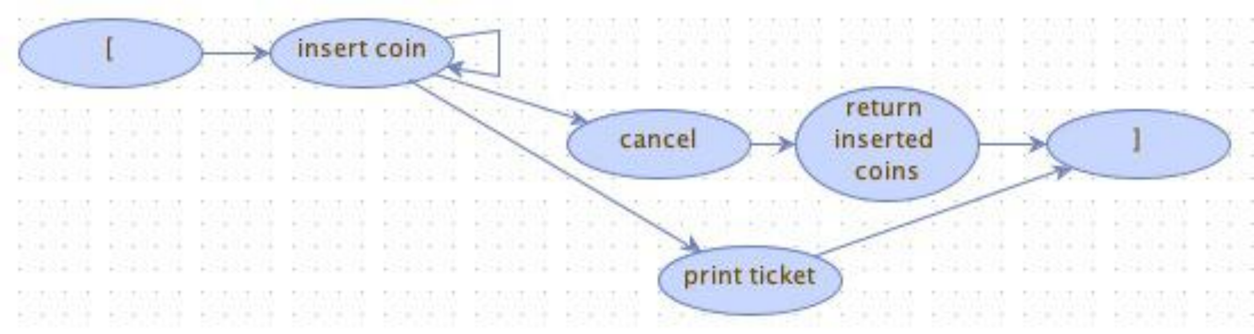




http://download.ivknet.de/index.php

State Machines to ESGs





Solution

SUT: simple_ticket_machine.mxe
****
Full Resolution Approach
****
Länge: 2
Nodes: 6
Edges: 7

Nodes (CPP): 6
Edges (CPP): 7

CES:
3: [, insert coin, insert coin, print ticket,],
3: [, insert coin, cancel, return inserted coins,],
No. of CES: 2
No. of Events: 6

FCES:
2: [, insert coin, return inserted coins,
3: [, insert coin, cancel, insert coin,
3: [, insert coin, cancel, cancel,
3: [, insert coin, cancel, print ticket,
3: [, insert coin, print ticket, insert coin,
3: [, insert coin, print ticket, cancel,
3: [, insert coin, print ticket, print ticket,
3: [, insert coin, print ticket, return inserted coins,
4: [, insert coin, cancel, return inserted coins, insert coin,
4: [, insert coin, cancel, return inserted coins, cancel,
4: [, insert coin, cancel, return inserted coins, print ticket,
4: [, insert coin, cancel, return inserted coins, return inserted coins,
1: [, cancel,
1: [, print ticket,
1: [, return inserted coins,
No. of FCES: 15
No. of Events: 42

Extended Modeling of GUI Components

- Input Contracts
  - use contracts to explicitly state expected behavior of both user and input component
  - use contracts to generate test cases
  - use contracts as test oracles
- Event Sequence Graphs (ESG) are used for modeling and validation of input component requirements.
- Contract supplemented ESG is developed to satisfy both of above requirements.

Extended Modeling of GUI Components

- Event Sequence Graphs
  - model the external behavior of the system

- Input Contracts
  - contracts are transformed into “decision tables”
  - links conditions (”if”) with actions (”then”) that are to be triggered depending on combinations of conditions (”rules”)

Input Contracts

- The contract notion is the key to describe input properties in precise terms.
- GUIs should be specifically designed to filter unwanted or unexpected input through input contracts.
- Model-based specification of input contracts is achieved through an input component model.

Input Contract Example

	Obligations	Benefits
User	(Must ensure pre-condition) Make sure that entered value for age is valid.	(May benefit from post-condition) Learn number of days lived or error message indicating what is wrong.
Input component	(Must ensure post-condition) Calculate and present number of days lived.	(May assume pre-condition) Give error message if value for age is invalid.

Input Component Model

- Input component model (Io, Dv, Ac, Co), where
  - Io is the set of GUI objects;
  - Dv denotes data variables;
  - Ac is the set of GUI actions;
  - Co denotes the component's input contract.
- Input component model augments ESG model.

Modeling input components

$\sigma = (Io, Dv, Ac, Co)$ with

- Io = {inputArea[Age], button[Enter], outputArea[Days]},
- Dv = {age, days},
- Ac = {exception, calculate}
- Co is as follows:

Modeling input components

- *Co* is

Conditions	R_1	R_2	R_3	R_4
pwv_age isTypeOf Integer	F	T	T	T
pwv_age > 0	-	F	T	T
pwv_age <= 150	-	-	F	T
A_1: Give error message	e	e		
A_{11}: Exception_{11}	X			
A_{12}: Exception_{12}		X		
A_2: Accept input			e	X
A_{21}: Exception_{21}			X	

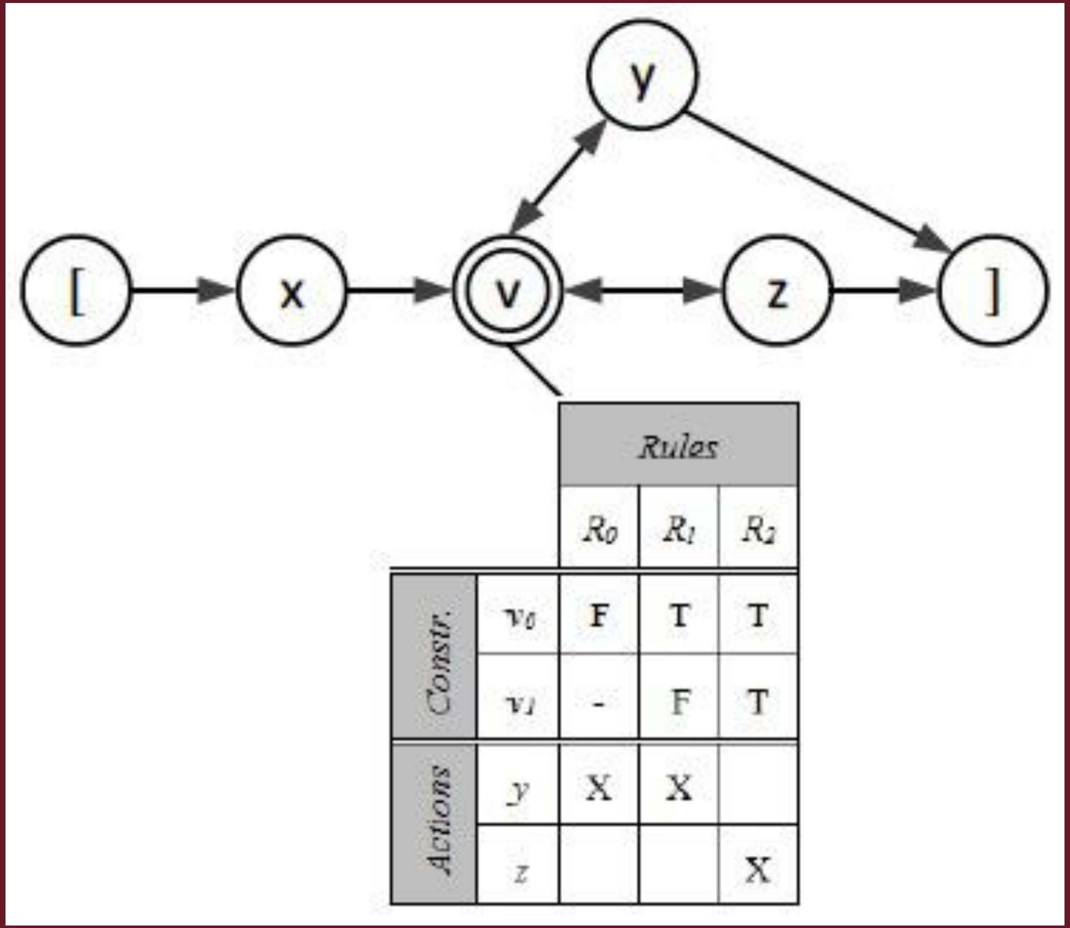
Exception_{11}: ERROR.AGE_NOT_INTEGER.
Exception_{12}: ERROR.AGE_LESS_THAN_OR_EQUAL_TO_ZERO.
Exception_{21}: WARNING.CHECK_AGE.

Decision Table Augmented ESGs (DT-ESGs)

- Decision Table Augmented ESG

- Decision Table

		Rules		
		R_0	R_1	R_2
Constr.	v_0	F	T	T
	v_1	-	F	T
Actions	y	X	X	
	z			X



DT-ESG Test Process

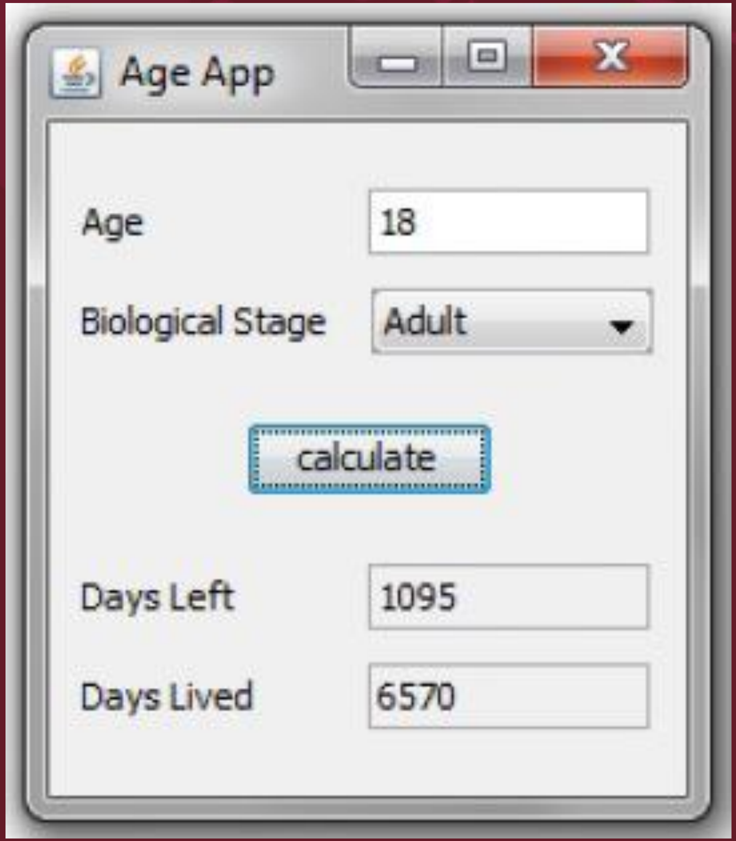
- generate the corresponding ESG
- cover all events by means of CESs
- foreach CES with decision tables do
- generate data-expanded CES using corresponding DT (input contract-based test case generation)
- apply the test suite to GUI
- observe GUI output to determine whether a correct response or a faulty event occurs

DT-ESG Example

- Age Application

Constraints	R_0	R_1	R_2	R_3	R_4	R_5	R_6
C_0: age is Type Of Integer	F	T	T	T	T	T	T
C_1: age > 0	—	F	T	T	T	T	T
C_2: age < 150	—	—	F	T	T	T	T
C_3: biologicalStage $=$ ADOLESCENCE and age $<$ adolescenceLB	—	—	—	F	T	—	—
C_4: biologicalStage $=$ ADULT and age $<$ adultLB	—	—	—	—	—	F	T
A_0: error/warning	e	e	e	e		e	
Exception$_{00}$	X						
Exception$_{01}$		X					
Exception$_{02}$			X				
Exception$_{03}$				X		X	
A_1: calculate					X		X

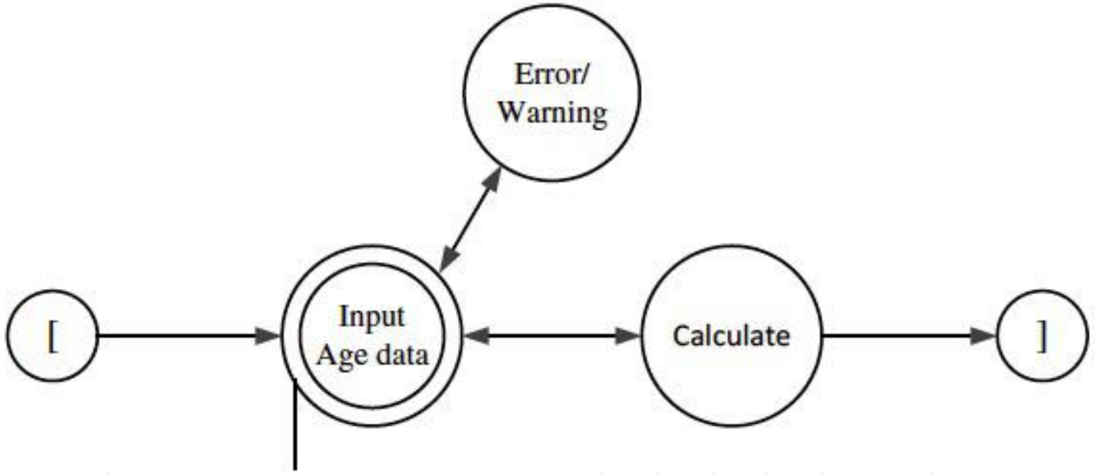
Exception$_{00}$: ERROR.AGE_NOT_INTEGER.
Exception$_{01}$: ERROR.AGE_LESS_THAN_OR_EQUAL_TO_ZERO.
Exception$_{02}$: WARNING.AGE_GREATER_THAN_OR_EQUAL_TO_150.
Exception$_{03}$: ERROR.NOT_PRIOR_TO_BIOLOGICAL_STAGE.



DT-ESG Example

- Age Application

[Input Age data(age:A,biologicalStage:Adult.> C0:F,C1:-,C2:-,C3:-,C4:-),
Error/Warning E00,
Input Age data(age:-1,biologicalStage:Adolescence.> C0:T,C1:F,C2:-,C3:-,C4:-),
Error/Warning E01,
Input Age data(age:200,biologicalStage:Adult.> C0:T,C1:T,C2:F,C3:-,C4:-),
Error/Warning E02,
Input Age data(age:18,biologicalStage:Adolescence.> C0:T,C1:T,C2:T,C3:F,C4:-),
Error/Warning E03,
Input Age data(age:7,biologicalStage:Adolescence.> C0:T,C1:T,C2:T,C3:T,C4:-),
Calculate,
Input Age data(age:25,biologicalStage:Adult.> C0:T,C1:T,C2:T,C3:-,C4:F),
Error/Warning E03,
Input Age data(age:18,biologicalStage:Adult.> C0:T,C1:T,C2:T,C3:-,C4:T),
Calculate]

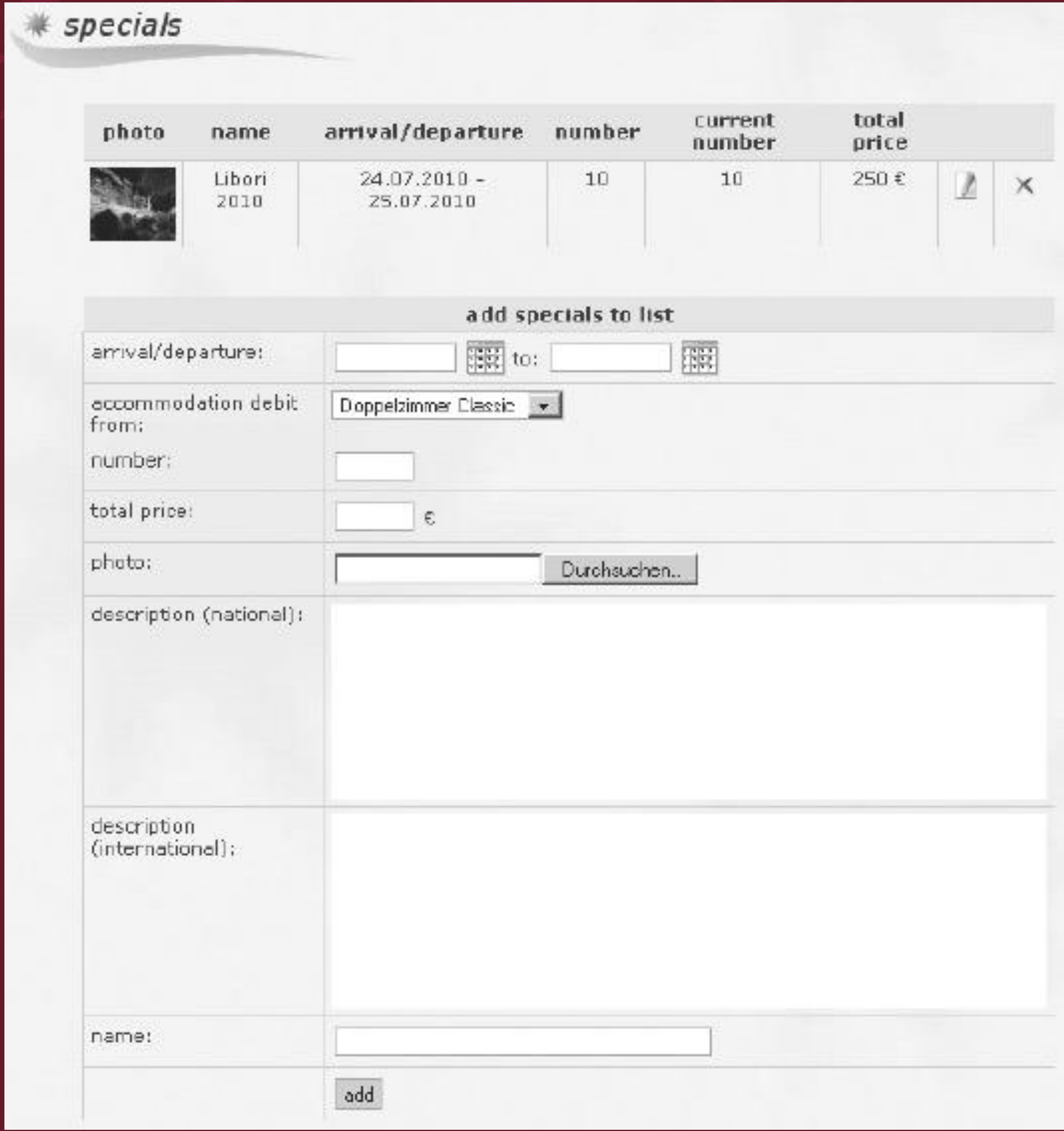


Constraints	R_0	R_1	R_2	R_3	R_4	R_5	R_6
C_0: age isTypeOf Integer	F	T	T	T	T	T	T
C_1: age > 0	-	F	T	T	T	T	T
C_2: age < 150	-	-	F	T	T	T	T
C_3: biologicalStage = ADOLESCENCE and age < adolescenceLB	-	-	-	F	T	-	-
C_4: biologicalStage = ADULT and age < adultLB	-	-	-	-	-	F	T
A_0: error/warning	e	e	e	e		e	
Exception$_{00}$	X						
Exception$_{01}$		X					
Exception$_{02}$			X				
Exception$_{03}$				X		X	
A_1: calculate					X		X

Exception$_{00}$: ERROR.AGE_NOT_INTEGER.
Exception$_{01}$: ERROR.AGE_LESS_THAN_OR_EQUAL_TO_ZERO.
Exception$_{02}$: WARNING.AGE_GREATER_THAN_OR_EQUAL_TO_150.
Exception$_{03}$: ERROR.NOT_PRIOR_TO_BIOLOGICAL_STAGE.

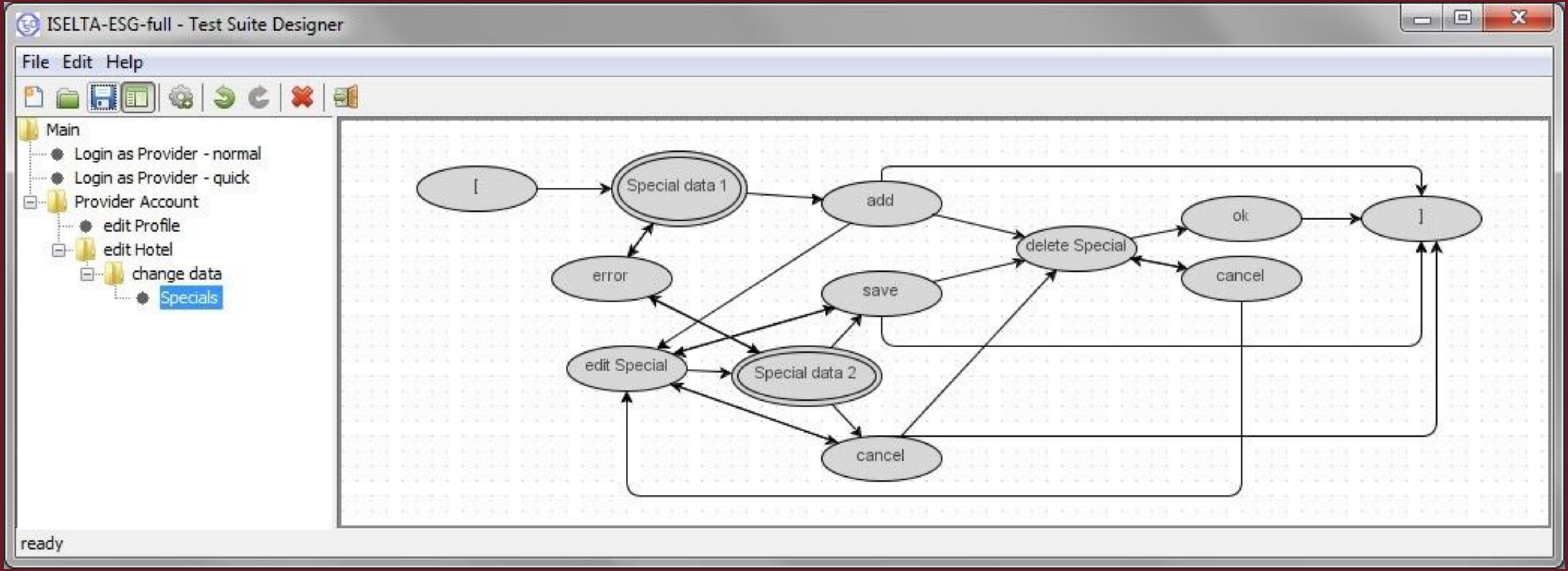
DT-ESG Example

- ISELTA Specials



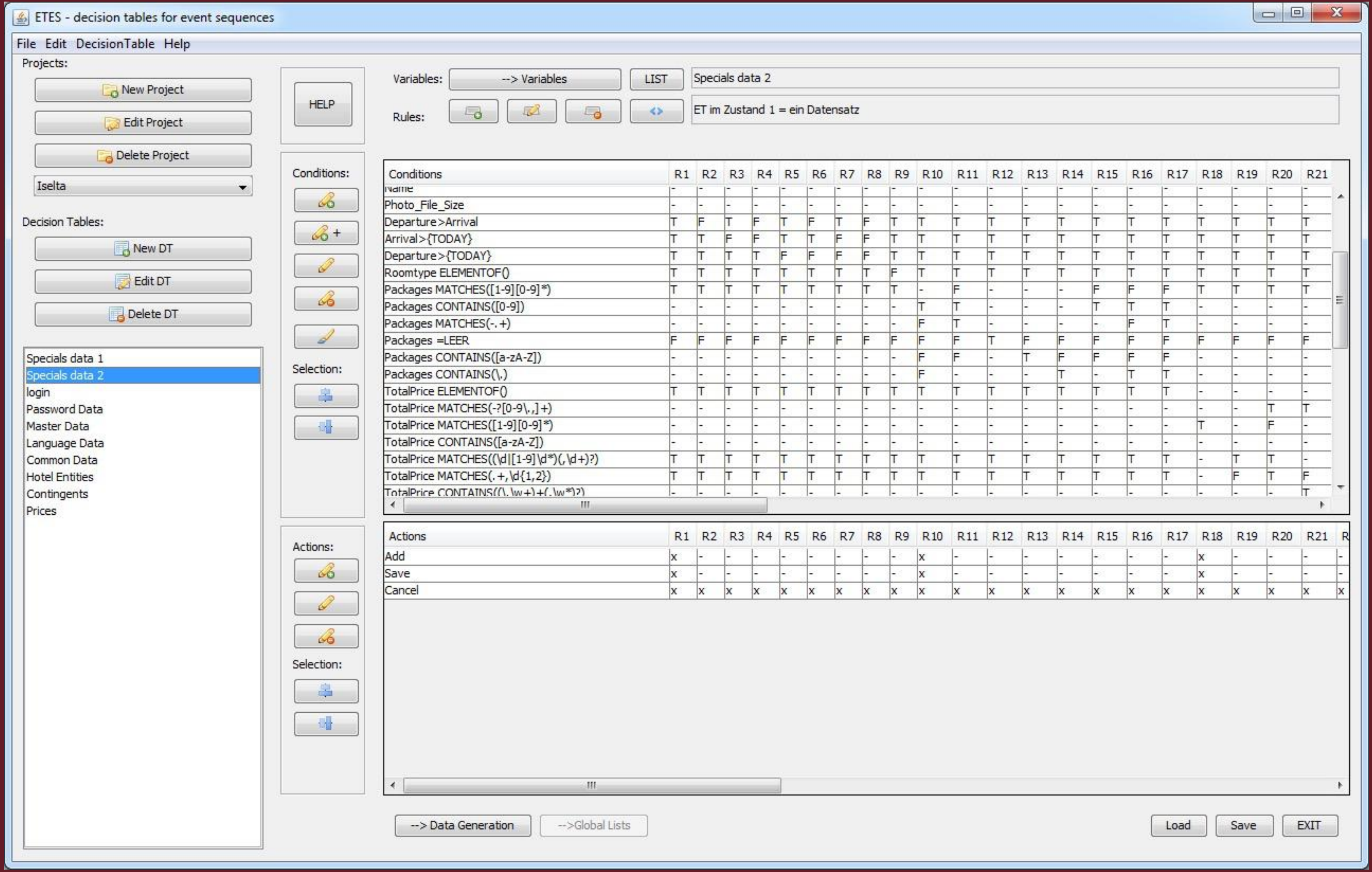
DT-ESG Example

- ISELTA Specials



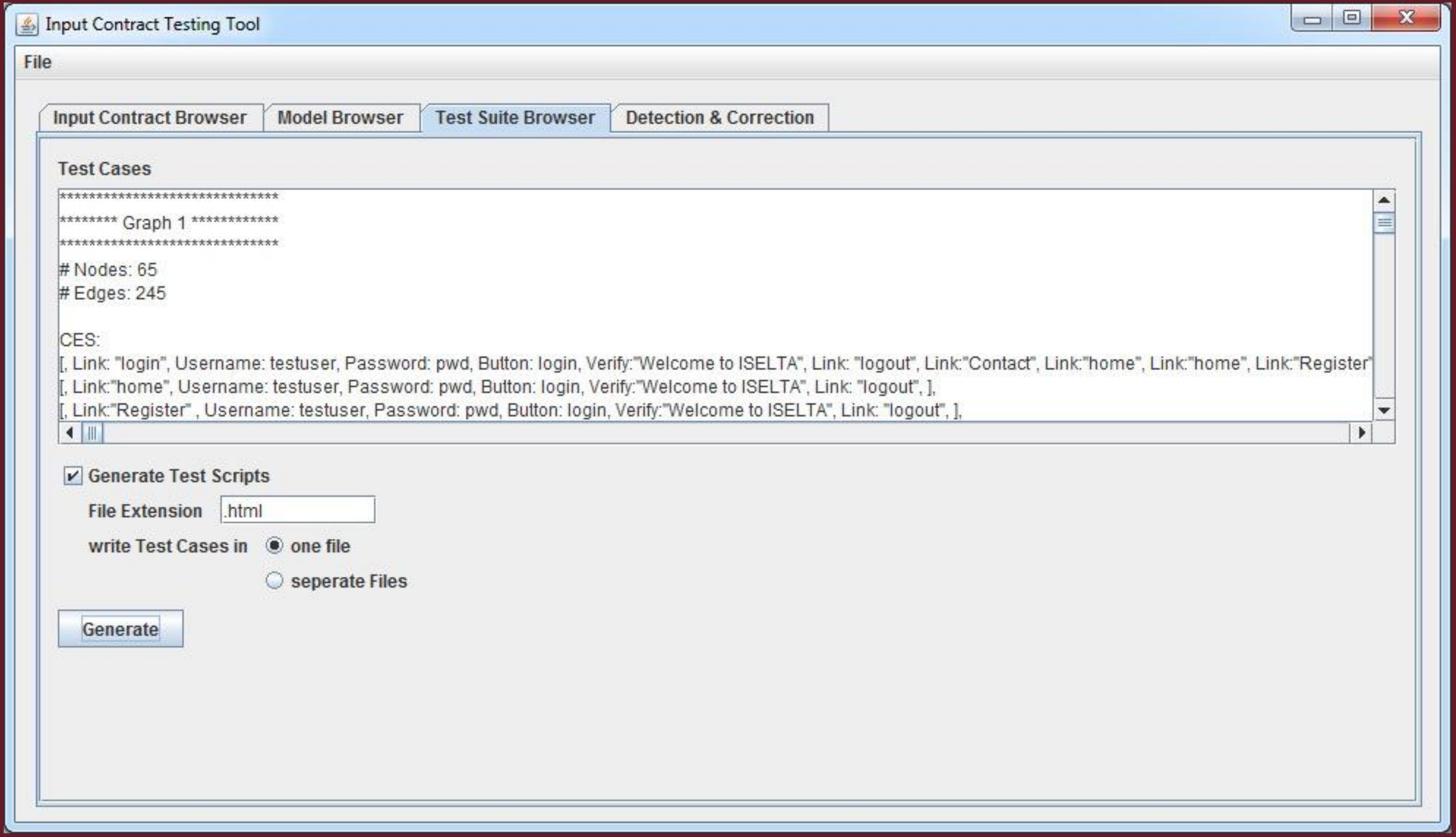
DT-ESG Example

- ISELTA Specials



DT-ESG Example

- ISELTA Specials



Lessons Learned

- Analysis of the results encourages the generalization that in the practice, pre- and post-conditions are not considered adequately, and thus counter-measure actions are neglected during software development.

- For existing software, tools such as the one introduced in this presentation are strongly recommended to prevent likely failures or undesirable situations that may occur because of deficiency control mechanisms in the software.

Lessons Learned

- If software is to be developed from scratch, then formal representation of input contracts are considerably useful for the correct implementation of specifications, as well as for the automation of software development and of software testing.

- Not only user interfaces but also component interfaces may be separated from business logic through input contracts, which may help both correct development and validation of the business logic part of the software under consideration.

Conclusion

- Model-based Event Sequence Testing with Input Contract Testing enables software test automation.
- Tools that automate software testing can be developed and practically can be used.
- Contract patterns and reusable contracts can add efficiency to test case generation.

Acknowledgement

- Prof. Dr. Fevzi Belli, Ph. D. , University of Paderborn, Germany
- Michael Linschulte, Ph.D. , University of Paderborn, Germany

References

- J.A. Whittaker, "Software's invisible users", IEEE Software 18.3, 2001, 84-88.
- F. Belli, "Finite-State Testing and Analysis of Graphical User Interfaces", 12th International Symposium on Software Reliability Engineering, ISSRE 2001, 2001, 34-43.
- A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage Criteria for GUI Testing", 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-9, ACM, 2001, 256-267.
- F. Belli, N. Nissanke, C. J. Budnik, and A. Mathur, "Test generation using event sequence graphs", Technical Reports and Working Papers, 2005.
- F. Belli, and C. J. Budnik, "Test minimization for human-computer interaction", Journal of Applied Intelligence, Springer, 26, 2, 2007, 161-174.
- F. Belli, M. Beyazıt, and A. Memon, "Testing is an Event-Centric Activity", The 6th International Conference on Software Security and Reliability (SERE), 2012.
- T. Tuglular, C.A. Muftuoglu, F. Belli, and M. Linschulte, "Model-Based Contract Testing of Graphical User Interfaces", IEICE Transactions on Information and Systems, Vol.E98-D, No. 7, pp. 1297-1305, July 2015.
- T. Tuglular, F. Belli, and M. Linschulte, "Input Contract Testing of Graphical User Interfaces", International Journal of Software Engineering and Knowledge Engineering, Vol. 26, No.2, pp. 183-215, March 2016.
- 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.