Towards Good Enough Testing: 
A Cognitive-Oriented Approach Applied to Infotainment Systems

Asem Eltaher
University of Braunschweig, Chair of Vehicle Electronic Systems, 38106 Germany
eltaher@ifr.ing.tu-bs.de

Abstract

This contribution outlines a cognitive-oriented approach to construct test systems that can “partially” imitate several cognitive paradigms of skilled human testers. For example, learning, reasoning, optimization, etc. Hence, a reasonable portion of the workload done by a human tester would be shifted to the test system itself. This consequently leads to a substantial reduction in the development time and cost; yet the test efficiency is not sacrificed.

1. Introduction

The process of testing and validation is a fundamental block of the development cycle; yet it is a formidable task especially for complex software products.

Related work dates to the notion of considering manual and automatic testing as complementary techniques. On the one hand, manual testing is costly, laborious, and possibly error-prone. On the other hand, “facing the fact that it is impossible to fully test a product” [1], skilled human testers work using rules of thumb rather than algorithmic methodologies. They possess an intelligent synthesis to design and adapt a significant range of test inputs under which a failure might arise. Then, they proceed further till additional testing does not change the test results. This process is defined in [2] as “good enough testing”.

One other approach is grounded on an automatic generation of the test cases from the designed specifications. This leads to their availability when the specifications are developed. Thus, the test process can be applied to partially designed systems, which can reduce the effort spent on expensive redesign [3]. However, the non-presence of the human tester in the testing environment may suffer from the absence of the effectiveness that an intuition of a skilled human tester brings to the test process.

Basically, the problem addressed here is to find a reasonable compromise between manual and automatic testing. This aims to establish a convergence between human-based testing towards an automatic test process to achieve sufficient shared benefits from both approaches.

Therefore, the idea is to design a test system that imitates several cognitive paradigms from a skilled human tester. Concurrently, the benefits from an automatic test process are still maintained. This leads to the test system’s ability to, not completely but partially, substitute the absence of a human tester during an automated test process.

Briefly, the adopted approach is to develop a test system that can learn an efficient test strategy by observing skilled human testers interacting with a Device Under Test (DUT). Then, the strategy learned must be optimized; using the lessons learned from other human testers, to perform additional testing that most likely triggers the hidden failures. Consequently, the strategy learned must be combined with other strategies from the past experiences to generate new test cases. Furthermore, the learning module has to serve the role of extracting the reusable task knowledge from observing Human-Machine-Interaction (HMI) to generalize the rules observed to be applied to other similar DUTs, or any further versions of the test suites.

2. Self-Learning by Observing HMI

2.1. Idea and Conception

HMI involves a considerable amount of cognitive hypotheses, e.g. learning, reasoning, optimization, etc. [4]. Hence, observing these hypotheses offers a promising approach to get an insight into the intelligence underlying the human behavior. To support the idea, the learning environment cannot focus solely on the intelligence underlying the human behavior. To support the idea, the learning environment cannot focus solely on the human tester, or on the DUT. Instead, it must span the boundary to observe both of them simultaneously. Fig. 1 shows the structure of the learning setup, in which a skilled human tester is let to test the DUT with no restrictions on the adopted testing art. Briefly, observing HMI during the test process is to serve the role of a training session for the developed test system.

During the ongoing phase of the training session, HMI has to be modeled as a preliminary step needed before the storage process takes place as shown in Section 2.2.
2.2. Situation-Operator-Model (SOM)

SOM in [5] is used to model the human tester-DUT interactions as a sequence of effects, which are described by the items scenes and actions. A scene is modeled by a situation whereas an action is modeled by an operator [5].

The item situation (S), shown in Fig. 2, is used to describe the observed state of the DUT, which consists of a set of characteristics (C) and relations (R). Each characteristic describes a definite part of the DUT, associated with a time-dependent parameter (P) that describes the current state of this characteristic. A relation (ri) is used to describe the inner connection(s) between different characteristics of the same situation. Fig. 2 shows also the item operator (O) that models the action(s) invoked by the human tester, which drive(s) the current situation of the DUT from an initial situation (SI) to a final one (SF).

Consequently, HMI is interpreted as an initial situation (Sj), an operator (or a meta-operator in case of multiple operators), and a final situation (Sk). The global structure of a situation changed by an operator that leads to another situation, defined in [5] as an experience, is shown in Fig. 2.

During the training session, an experience is defined, if the state of the DUT is changed, or a certain time period is elapsed. Then, the knowledge base saves the defined experience and the current final situation is defined as the initial one for the next experience, etc. I.e. SOM models HMI as a sequence of experiences. Furthermore, SOM offers significant advantages over other modeling techniques like Finite-State-Machine (FSM), or Unified Modeling Language (UML), which are described in [6].

2.3. Knowledge Base

The knowledge base, or “mental model” [5], formulates an internal description of the testing know-how adopted by the human testers.

![Figure 1. Learning by observing human testers.](image)

![Figure 2. A situation changed by an operator [5].](image)

Additionally, a learning system must not only develop its mental model, but also refine it. Therefore, it has been enriched with a 2D visualization module to facilitate the human feedback process. This process involves the ability to not only acknowledge the consistency of the experiences learned, but also to alter them. The former case is triggered, if the test system observed inconsistent test cases, e.g. the same test case with different results due to a non-observable human error. On the other side, altering the knowledge base is demanded, if the DUT provided a wrong reaction during the learning phase.

Obviously, learning from a sole human tester may suffer from the cognitive biases that have been found in humans. To overcome this limitation, a framework to learn from multiple human testers is implemented. During learning, the test system discriminates the experiences observed into simple (deterministic) and compound (non-deterministic) test cases. In simple test cases, the relationship between the action and reaction is a one-to-one correspondence. On the other side, compound test cases involve a many-to-many correspondence. Thus, the test system must apply clustering heuristics to reason about, which action has triggered which reaction.

3. Supervised Clustering

3.1. Idea

Supervised data clustering serves the role of partitioning a data set into clusters, or classes. This leads to the assignment of the similar data to the same cluster whereas dissimilar data should belong to different clusters [7].

The potential of the developed test system to assign an operator to its corresponding affected characteristic is referred in this work as learning the inner structure of the DUT. Rule-based reasoning (RBR) in [8] is used in case of simple test cases whereas case-based reasoning (CBR) in
3.2. Supervised Clustering using RBR

RBR is considered as the ability to reason using a prior experiential knowledge acquired from human experts [8]. In our case, the prior experiential knowledge involves the assignment of the operator to the characteristic with the triggered parameter (see Fig. 3a). Then, the defined relation is stored in the knowledge base to avoid performing the reasoning algorithm in case of similar experiences. In case of compound test cases, RBR claims a situation of a non-detachable experience and CBR component is activated.

3.3. Supervised Clustering using CBR

CBR is based on an analogical transfer approach, in which a new problem is solved by finding a similar past case and reusing it in the new case [8]. Within the scope of this paper, the similarity of two experiences is defined as:

**Definition:** Two experiences ([S₁₁, O₁, S₁₂] and [S₂, O₂, S₁₃]) are similar, if the initial situations coincide and the operator set of one experience is a subset of the other one. I.e. S₁₁=S₁₂ and O₁ C O₂. Therefore, CBR can be used in the following fashion: a) searching the knowledge base for a similar experience, e.g. Eₖb, b) integrating the past experience into the current observed one, e.g. Eₒb, and c) saving the new experience in the knowledge base for future similar cases (see Fig. 3).

Supervised clustering is adopted because of its strong coupling to the idea of learning from the past experiences, which definitely fits to our problem. However, it is planned to compare it with other learning theories, e.g. computational learning theory in [8], etc. to evaluate the hypothesis of the adopted learning technique.

3.4. Optimization and Generalization

Obviously, a learning test system would be ultimately of a little value; I) if it just blindly applies the strategy learned to the DUT used during the learning phase in case that additional testing is needed, or II) if is not able to generalize the strategy acquired by observing a human tester to be used in testing similar DUTs.

Therefore, the contribution of the learning phase into the testing procedure has to be divided in two distinct cases; a) further testing on the same DUT used in the learning stage, but flashed with a newer software version, which is denoted in this work as the optimization phase, b) the use of the strategy observed to test other similar DUTs, i.e. different hardware and software versions, which is denoted here as the generalization phase.

In the real world, case (a) takes place quite often (e.g. typically once per 2-4 weeks), in which the same hardware products are flashed with a new version of the software either to test a new feature, or to test if the bugs from the previous version are fixed. Whereas case (b) takes place in case that the provider decided for some reasons to change the manufacturer of one, or more component(s) of the whole system to be tested. If an infotainment system is considered, an example of case (b) would be an economic-based decision from the provider to change the manufacturer of the audio amplifier while keeping the other components of the whole infotainment system unaltered as the navigation system, radio, telephone, CD-player, etc.

In the optimization phase, i.e. case (a), the test system establishes a memory-based reachability graph that consists of all the configurations and possible paths that are reachable from the initial configuration. Then, it will adopt a commonly used criterion of test coverage, which is to test each edge at least once. Additionally, based on the reasoning module, the test system can learn the inner structure of the DUT and generate global specifications of the overall operational manner of the DUT. Then, test cases generation techniques based on model-based specifications as in [9] can be applied to generate further test scenarios that have not been observed before. I.e. the test system searches in the generated specifications, described in State-Flow-Diagram, for a sequence of test inputs that has not been triggered before by the observed human testers.

In the generalization phase, i.e. case (b), behavioral primitives’ approaches in [8] shall be applied to facilitate the issue of generalizing the testing rules to formulate test oracles that can be used in testing similar DUTs. Therefore, the need to build a learning environment for each DUT separately is avoided.

For example, in the training phase whereas an audio amplifier is tested as a component of the target infotainment DUT, the test system does not learn typical numerical values of the volume signal in case the action Set_VOL is invoked by the human tester. Rather, it learns a relationship between the incremented/decremented percentage of the
volume’s signal with respect to the number and the direction of the button’s turns. This rule can be applied further to test the volume function of any future software versions of the same audio amplifier, or even a new hardware device in case the manufacturer has been changed. However, a behavior’s tolerance must be defined due to different software versions, or even different manufacturers; nevertheless the testing rule learned in the training session is still applicable.

4. Conclusion and Outlook

This contribution outlines a novel approach to realize a test system that can partially imitate the intelligence of skilled human testers. The functionality of the proposed framework can be interpreted into four modules; learning, reasoning, optimization, and generalization.

The proposed framework of testing systems is more effective than the conventional test systems in several innovative aspects. Learning a sequence of test cases by observing human testers eliminates the time and energy devoted to write script-based test scenarios.

Besides, the reasoning paradigm can identify the inner structure of the DUT, which leads to an automatic generation of the operational specifications that describe the global behavior of the integrated infotainment system. Then, test cases generation techniques based on model-based specifications can be applied to generate further new test scenarios that have not been triggered before.

Additionally, the theory of optimization can lead to a substantial reduction in the traditional execution time of the manual test cases, if a new version of the software is available whereas the gleaned testing know-how is still maintained from the past lessons learned.

Finally, the generalization module offers a constructive idea to apply the strategy learned to similar DUTs, which definitely saves a tremendous amount of time and effort devoted in testing newer, or alternative versions of the devices due to any migration plans by the providers.

Learning by observation and the reasoning modules have been implemented and the initial results are shown in [10]. Whereas the optimization and the generalization modules are the key elements of the future work.

The central issues of the approach’s validation are to evaluate the system’s performance against several aspects. For example, in case of additional testing of the same DUT, the efficiency of the self-generated test strategy in terms of the execution time and states coverage criteria, the selective ability of the framework to generate -based on the lessons learned- test scenarios that most likely fail, etc. In case of using the developed framework in other test suites, a significant metric would be, in which extent the strategies learned can be generalized to be applied to other DUTs. In addition to how far the test oracle generated in the training session can properly evaluate the reactions of the new DUT within an allowed tolerance.

Acknowledgments

I owe a great deal of gratitude and appreciation to Professor Thomas Form and Professor Markus Maurer for their very fruitful contributions to my PhD work, in addition to their productive guidance and help. Furthermore, I am definitely indebted to Dr. Mohamed Ayeb who sacrifices a great deal of continued support and encouragements to bring my work to the next level.

References