

A Case Study of Interactive Development of Passive Tests

Daniel Flemström
RISE SICS Västerås AB
Västerås
daniel.flemstrom@ri.se

Thomas Gustafsson
Scania CV AB
Södertälje
thomas.gustafsson@scania.com

Avenir Kobetski
RISE SICS AB
Stockholm
avenir.kobetski@ri.se

ABSTRACT

Testing in the active sense is the most common way to perform verification and validation of systems, but testing in the passive sense has one compelling property: independence. Independence from test stimuli and other passive tests opens up for parallel testing and off-line analysis. However, the tests can be difficult to develop since the complete testable state must be expressed using some formalism. We argue that a carefully chosen language together with an interactive work flow, providing immediate feedback, can enable testers to approach passive testing. We have conducted a case study in the automotive domain, interviewing experienced testers. The testers have been introduced to, and had hands-on practice with a tool. The tool is based on Easy Approach to Requirements Syntax (EARS) and provides an interactive work flow for developing and evaluating test results. The case study shows that i) the testers believe passive testing is useful for many of their tests, ii) they see benefits in parallelism and off-line analysis, iii) the interactive work flow is necessary for writing the testable state expression, but iv) when the testable state becomes too complex, then the proposed language is a limitation. However, the language contributes to concise tests, resembling executable requirements.

CCS CONCEPTS

• **Software and its engineering** → *Domain specific languages; Integrated and visual development environments; Application specific development environments; Software testing and debugging;*

KEYWORDS

passive testing, case study, content analysis, test language, test tool

ACM Reference format:

Daniel Flemström, Thomas Gustafsson, and Avenir Kobetski. 2018. A Case Study of Interactive Development of Passive Tests. In *Proceedings of 5th International Workshop on Requirements Engineering and Testing, Gothenburg, Sweden, June 2018 (RET'2018)*, 8 pages. https://doi.org/12.345/123_4

1 INTRODUCTION

Testing is a resource hungry undertaking where organizations strive for improved efficiency. Most testing is active testing where a tester is *actively* stimulating the system under test (SUT) while observing its response to form a *test verdict*. Usually, the test verdict verifies

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
RET'2018, June 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 123-4567-12-123/12/12...\$15.00
https://doi.org/12.345/123_4

the implementation's correctness of a linked requirement. Passive tests are *independent* from each other and the test stimuli, and this separation of concerns opens up for evaluation of tests in parallel, but also generating inputs that cover as large input space as possible.

One way to reason about passive tests is the Guarded Assertions (GA) approach [7], where the test is split into a *guard*, stating the testable state of the SUT, and the *assertions*, stating the expected response of the SUT in the testable state. The high level of formalism usually used to express a GA might be a limiting factor for industrial practitioners [1, 16]. Pattern-based requirements specification methods, like Easy Approach to Requirements Syntax (EARS) [12], have been proposed to address these issues in the requirements engineering field [8, 12, 15].

Tool support can be one way to introduce passive testing to practitioners. One such toolbox centers around an interactive work flow where the test engineer, using the extended EARS formalism, describes the test logic using the GA approach [4]. While the test logic is being edited, the toolbox shows plots of relevant guards and assertions, based on a pre-loaded input-output trace of the SUT.

In order to investigate in what ways the toolbox work flow and the language affect the testers, we chose to conduct semi structured interviews with testers from five different test groups at a heavy truck manufacturing company. Each group is using Hardware-In-the-Loop (HIL) as a means for performing the testing, and are thus heavily using automated testing. Furthermore, these five groups represent test levels ranging from Electronic Control Unit (ECU) to integration testing of the complete vehicle electrical system.

The results of these interviews are presented in this paper. Section 2 shortly presents the theory behind the toolbox, as well as the tool itself. Section 3 describes the design and realization of the case study, including validity considerations. Section 4 collects the results of the interviews, Section 5 discusses related work, while Section 6 concludes the paper.

2 BACKGROUND

This section contains a background of the toolbox mentioned above, including a description of the guarded assertions approach, the pattern-based language used by the toolbox.

2.1 Guarded Assertions

A *guarded assertion (GA)* [7] is a logical construct, suitable for passive testing, consisting of two parts: i) the *guard* specifying the state of the SUT where the test shall commence, and ii) the expected responses, i.e., the *assertions* of the SUT when it is in a testable state. We may consider the following example: When a reverse gear is engaged on a vehicle with a turned on electrical system, the rear lamp shall be enabled. The guard is an enabled electrical system and engaged reverse gear. The expected response is that the rear lamp is on.

The above described testing can be denoted *passive testing*, since the test never changes the state of the SUT, but also a requirement, because requirements may not consider individual input sequences.

2.2 Timed Easy Approach to Requirements Syntax (T-EARS)

Easy Approach to Requirements Syntax (EARS) [12] is a semi-formal way of expressing requirements using a set of pre-defined keywords. The keyword set, based on best practices from an industrial experience, is intentionally kept short and simple so as to reduce the threshold needed to start using EARS. By imposing some other restrictions on the format, like using logical propositions and known parameter names instead of natural text descriptions of them, EARS-expressions can be converted into a machine readable format, significantly enhancing the rigor of requirements, as compared to natural language formulations [6]. The basic EARS templates are:

- ubiquitous: requirement is always active
- keyword *when*: required response to a triggering event
- keyword *while*: required response in a specified state
- keyword *where*: applicable only if a certain feature is included
- keyword *if... then*: required response to an undesirable triggering event

A limitation of EARS when it comes to testing real-time systems is its lack of templates for describing timing information. For this reason, EARS was extended into timed EARS (T-EARS) [4] using the following keywords:

- keyword *within*: the time within which a condition must hold
- keyword *for*: minimum time that a condition must hold
- keyword *after*: relative order of two events

In addition, T-EARS has some convenience constructs, such as `abs(sig)` - the absolute value of a signal, and `even(sig)` - the time intervals when `sig` has been high an even number of times. It is rather straightforward to extend T-EARS with other constructs.

2.3 Toolbox

The SAGA toolbox [4] shall enable a test engineer to work efficiently on three aspects of test development and test result analysis: i) developing a GA, ii) analysis of a test result of a GA, and iii) developing SUT stimuli. The toolbox has a web based interface centered around an interactive work flow in the following way:

- loading a trace from the SUT,
- a listbox presenting available signals in the trace,
- a text editor supporting the T-EARS language with syntax highlighting and auto-completion of signals,
- a set of plots representing guards, with visual highlighting of the time intervals where the guard is valid, and
- plots of assertions, with visual highlighting of the time intervals where the assertions pass or fail.

As soon as the T-EARS expression is changed, the plots are updated to accurately represent the new GA. This gives an interactive round-trip, where the test engineer can study the plots, zoom if needed, and update the GA if needed. The tool can also be used as a query language on the trace.

3 CASE STUDY DESIGN

This section contains the objective and research questions, as well as a description of the context, preparation and data collection, the analysis procedures and finally an analysis of the threats to validity and the countermeasures undertaken.

3.1 Objective and Method Selection

We argue that tool support is important for enabling passive testing to practitioners, thus, certain design decisions have been taken in the SAGA toolbox presented in Section 2. The objective of this study is to assess how well the GA approach, the T-EARS language, and the interactive work flow enable passive testing in an industrial setting. The objective is broken down into three research questions:

- RQ1. What are the potential benefits and drawbacks of the GA approach in an industrial testing setting?
- RQ2. What are the potential benefits and drawbacks of using the T-EARS language to model GAs?
- RQ3. What are the potential benefits and requirements on an interactive tool for the development of GAs?

To collect data for answering the research questions, we chose to perform an exploratory case study, based on interviews with software test practitioners within the context of automotive systems testing. Since benefits and drawback are composed of a multitude of aspects that may be intertwined and possibly expressed indirectly, the interviews were subjected to a content analysis [5].

3.2 Context

The case study was carried out at Scania CV AB, one of the world's largest manufacturers of heavy trucks, buses, and engines. Automated testing is performed, using SIL and HIL, on all test levels up to and including testing of the complete-vehicle electrical system. For regression testing purposes, the testing is automated meaning that batches of tests are automatically executed and the test results and logs are collected for reporting and analysis.

At Scania, there are test groups that focus on testing specific parts of the electrical system. Each group is using a test automation framework for setting up their specific test environment, providing an interface for test scripts, and a means for producing logs and test results. The tests are implemented using contemporary programming languages like Python and C#. The test scripts are thus regular computer programs undergoing a normal development cycle with testing, debugging, version control, review, etc.

3.2.1 Participants. The focus of this case study was on test groups using either SIL or HIL environments, since these were deemed to benefit the most from the toolbox and the GA approach in the short term. In total, there are at least 50 employees working in these groups at the company. One experienced test engineer was chosen from each of these groups, resulting in five interviewees doing the following testing: ECU testing of body communication interface, ECU testing of cab systems, ECU testing of gearbox control system, system-level testing of engine management system, and, complete-vehicle electrical system integration testing. Moreover, one test-run interview was made with an experienced test engineer from the complete-vehicle electrical system integration testing group, resulting in six interviews in total. Table 1 shows a summary of the participants in the case study.

Table 1: Interviewee Summary

| Interview | | | Department | | | Participant | | |
|-----------|----------|--------------|------------|-----------------------|--|-------------------------|------------------|--|
| Id | Duration | # Statements | Id | Test Level | Roles | Selected TC Size | Years of Testing | |
| 1 | 1h50min | 82 | A | ECUs Test | SE ¹ , T ² , TL ³ | Fairly simple | 16 | |
| 2 | 1h13min | 124 | B | Complete Vehicle Test | T | Fairly simple + simple | 3 | |
| 3 | 1h00min | 109 | C | ECUs Test | T | Many small simple tests | 4 | |
| 4 | 1h48min | 70 | A | Complete Vehicle Test | T | Fairly simple | 12 | |
| 5 | 1h09min | 112 | D | ECUs Test | SE, T, CGL ⁴ | Fairly simple | 14 | |
| 6 | 1h29min | 106 | E | ECUs Test | SE, T | Fairly simple | 8 | |

¹Senior Engineer, ²Tester, ³Test Leader, ⁴Competence Group Leader

3.3 Preparation and Data Collection

In this section, the preparation and data collection procedures of the case study are presented, in accordance with the general guidelines found in [17]. Figure 1 shows an overview of this process that consists of preparation, trial, and finally the interview phases.

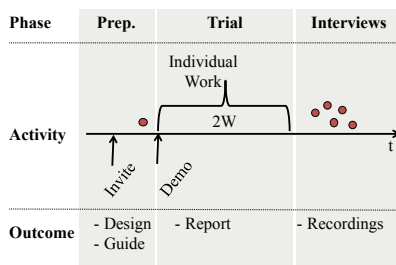


Figure 1: Data Collection Process

3.3.1 Preparation Phase. The preparation phase constitutes several activities. An initial understanding of the case context was achieved by studying available test cases (TC) and having informal discussions with some of the testers about TC organization. This phase was ongoing during a rather long period of time.

An interview guide was constructed using open-ended main questions as well as follow-up questions. Where we could foresee the need, a few probing questions, e.g., “why do you”, were added to the questionnaire. The questions in the guide were divided into three content areas (Approach, Language and Tool), reflecting the scope of the research questions in Section 3.1. Further, as a warm up section, the guide started with contextual questions regarding background information about the interviewee, such as experience and current ways of working, meant to facilitate the result comparison.

This interview guide was tested in a test-run to allow adjustments before the actual interviews were performed. Since no changes were necessary, the test-run was included in the final data set.

3.3.2 Trial Phase. The selected participants of the case study received an invitation letter, describing the objectives of the study as well as the data collection procedures and that the interviews would be recorded and anonymized. Further, each participant was asked to select test cases of varying complexity from their own domain. The test cases should also have been automated in their current test automation framework, to allow the participants to make comparisons. The purpose of the trial phase was to allow the participants to gain hands-on experience of the toolbox and its underlying concepts, while working on their own test cases.

As shown in Figure 1, the trial phase started with a demonstration session, when the concept of guarded assertions, T-EARS, as well as basic tool handling skills, were presented. The demonstration was carried out on one occasion, with all participants together. Next, the participants were expected to work with their selected test cases and the SAGA tool for a period of at least two weeks calendar time, depending on when their interview was scheduled. The duration was chosen to minimize conflicts with the testers normal work tasks. The participants were encouraged to take notes during this phase in an individual experience report.

3.3.3 Interview Phase. The participants were individually interviewed in Swedish by two researchers, one performing the interview and the other taking notes. The interviews were also recorded. During the interview, the interview guide was the main instrument, but the experience report was also discussed. The target time of the interviews was set to 1-2 hours.

3.4 Analysis Procedures

This section describes the analysis procedures as illustrated in Figure 2. Content analysis, as described by Graneheim and Lundman[5], was carried out on the interview material. They divide the analysis into manifest content analysis that describes “what” the participants are telling, and latent content analysis, where an underlying meaning is sought “between the lines”. Although the participants in the study have vast experience in software testing, their experience with the tool is limited in comparison. As a consequence, we strive for keeping the analysis more manifest than latent.

3.4.1 Transcription. The recordings were transcribed at a verbatim conversational level. Nuances in language or body language were not considered. To anonymize the results, each interview was assigned a random number (see Table 1). The outcome of this step was one file with raw text for each interview.

3.4.2 Familiarization. The full transcript of each interview was read several times to get an overview of the material. This resulted in a main-takeaways section, added to each interview to communicate the gist between researchers not taking direct part of the analysis.

3.4.3 Split. The transcripts were split into *meaning units*. A meaning unit typically corresponds to one sentence. Each such meaning unit was assigned a unique row-id in an Excel sheet. Approximately 4800 such meaning units were extracted from the transcript. The outcome of this step was an Excel sheet as: <row-id, interviewee-id, text>, containing all meaning units.

3.4.4 Extraction. Not all meaning units contained useful information, while others were excessively wordy or contained several pieces of information. To mitigate this and to reach a manageable

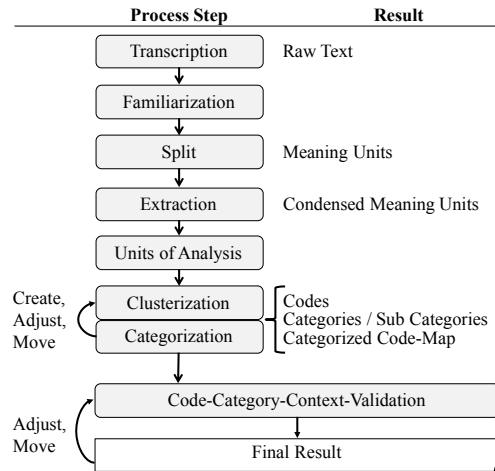


Figure 2: Overview of the analysis process.

level of analysis, the contained information was extracted from the original list of meaning units into *condensed meaning units*. The output of this stage was the same Excel sheet as in the Split step, with the additional condensed meaning unit column.

3.4.5 Units of analysis. To form manageable units of analysis, the condensed meaning units were sorted into content areas as described in Section 3.3.1. Each content area was then analyzed separately in the steps clusterization and categorization below.

3.4.6 Clusterization (coding). As depicted in Figure 2 clusterization and categorization, were undertaken iteratively. During the categorization of the condensed meaning units, units with the same meaning were grouped together and tagged with a code. Examples of such codes are “[Test Mgmt System] {217,218,2404,...4114}” and “[Difficult w-o recorded data]{491,960,3404...3379}”. The numbers represent the units with the same meaning. The outcome of this step is a set of categorized codes.

3.4.7 Categorization. Categories and sub-categories were created in an iterative bottom-up approach. The codes were moved around between categories and sub-categories and new categories emerged until no further new categories or movements could be motivated. This step also included moving of codes that belong to another content area. The result of this step was a tree like category-code-map that served as a mind-map for the results section.

3.4.8 Validation. When splitting the content and rearranging the pieces there is a risk that some of the pieces end up in a completely different context. To prevent such mistakes, each code-usage needs to be validated against the original text. The categories and contained codes were added to the results section of this paper, and the row-ids of each code were expanded to <row-id, interviewee-id, condensed meaning unit> and validated to the original meaning units (typically in a window of 5 before and 5 after). This produced a validated result list where misclassified codes and out of context condensed meaning units had been addressed.

3.5 Validity

When performing a case study there are several threats to validity [17]. *Construct validity* is concerned with whether the method and data sources are relevant to answering the research questions.

There is a risk that the result is biased by the choice of interviewees. As a countermeasure, the interviewees were selected from different departments. To mitigate the risk of researcher bias in the interview answers, the questionnaire was carefully designed to not reveal any pre-assumptions or expectations. However, there is always a risk when demonstrating the tool and concept, that certain features are given more attention than others. Since one of the researchers is employed at the case organization, there is a potential risk that this could influence the testers answers. To mitigate this, the two researchers not employed at the case organization, performed the questioning. Another risk is that the researcher finds the answers he expects. To mitigate this, the transcripts were checked against the recording by another researcher.

One threat to *internal validity* is that the interviewees have long experience of their current testing framework compared to the GA approach. Another threat is that the SAGA tool is under development and is not as mature as their current tools.

As in all case studies, there are several threats to *external validity*. Since we have studied only one organization, there is a risk that the results are not generalizable. To mitigate this risk we have selected the interviewees from different departments with a vast experience from different organizations. Further, the context, for which the study is valid, has been described as thoroughly as possible. Since the study is exploratory, using content analysis as a primary analysis method, there are several threats to *reliability*. One risk is that another researcher would have coded the transcripts differently. This was mitigated by having a second researcher validating a random set of the encoded meaning units. Another threat is that the categorization is dependent on the researchers pre-assumptions. As countermeasure, the category maps and encoding were presented to and discussed with the other researchers.

4 CASE STUDY RESULTS

The results of the interviews are structured along the content areas. Within each content area, the results are presented following identified categories of meaning units, and underlined with quotations, where appropriate. Note that these quotations refer to generalized translations of key findings and are thus not direct quotations from the interviewees. Still they capture the gist of the original statement in the raw data, e.g., “*T-EARS can be used for avoiding misunderstandings.*” [i3], where [i3] refers to the *id* column of Table 1.

4.1 GA Approach

The collective results of this content area serves the purpose of answering RQ1. - “*What are the potential benefits and drawbacks of the GA approach in an industrial testing setting?*” Two of the identified categories (*Potential* and *Risk*) directly contributes to answering the question. However, the analysis also revealed the category *Applicability and limitations*, with subcategories describing when and why the potential or risks can be expected.

4.1.1 Potential. This category contains the properties that contribute to the potential of the GA approach, with each property represented by a subcategory.

Efficiency: The passive approach to testing leads to increased parallelism in the sense that it allows several requirements to be verified at the same time. This is perceived as more efficient by the interviewees. “*It will be more efficient if I can run my state machine once*”

465 and run all tests [in parallel] instead of running it once, checking
466 this, running again, checking that, and so on.” [i6] In addition, a
467 GA is evaluated as soon, and as often, as the system has reached a
468 testable state. According to the testers, this often happened sooner
469 than expected, which indicates that the required test sequence may
470 be shortened. “Another benefit we discovered with our present test
471 automation framework was that we shortened our HIL execution time.
472 Because, often, the expected system state occurs earlier than the tester
473 guessed. Sometime this could mean several hours on a test suit.” [i3]
474 **Effectiveness:** As a consequence of the continuous execution of GAs,
475 a requirement may be tested several times, which is an advantage
476 compared to current testing where tests are executed once. “If we
477 had some overall tests during the entire test session, we might catch
478 more than today.” [i6] Improvements with respect to regression
479 testing were mentioned: “For regression test purposes, it may be very
480 useful to have a set of selected test cases waiting for their active state
481 in the background while you test other things at depth.” [i1]

482 **Diversity:** The ability of running tests off-line on a great variety
483 of log-files was mentioned as a great potential of the approach.
484 Examples given were old logs, logs from other test cases, other
485 people’s logs and field vehicle logs. This enables running a modified
486 test without having to re-run a HIL execution, which could be used
487 for deep-analysis leading to an increased effectiveness. “Extremely
488 good if you can test off-line [on different input data] when you have
489 changed a test.” [i4]

490 **Realism:** The possibility to analyze logs from field tests directly leads
491 to more realistic tests. Altogether this contributes to an increased
492 confidence in the test results and that more bugs are believed to be
493 found. “Given that we can analyze logs from field tests we get more
494 reliable and realistic tests.” [i5]

495 **Clarity:** Another positive effect of the way tests are specified in
496 the approach, is that the tests are perceived as more “clear”. One
497 reason is that the stimuli sequence is not mixed into the test case.
498 Another reason is that the guard thoroughly describes the testable
499 state of the system and is clearly separated from the assertion part.
500 Finally, evaluating test cases in parallel opens up for having smaller
501 test cases, ideally covering one requirement at a time, which also
502 contributes to increased clarity. “It is much more clear when you have
503 the actual system state together with what you want to compare.” [i2]

504 **Robustness:** Having smaller test cases, as mentioned in *Clarity*, also
505 contributes to increased robustness: “I think that, just the fact that
506 we are forced to write smaller tests with fewer inputs and a well
507 defined scope for one thing at a time, instead of testing many things
508 at the same time would help us to get more robust testing.” [i6]

509 **Reuse:** The testers considered it beneficial to reuse the GA logic
510 from lower integration levels, even though the test automation
511 frameworks usually differ and the logic would have to be combined
512 when moving up in the integration chain.

513 **System understanding:** The ability of fine tuning expressions by
514 applying them on an off-line log was also considered as a means
515 of gaining understanding of both the requirements and the system
516 behavior. This is further emphasized by the observation that in
517 order to thoroughly describe the testable state, the tester is forced
518 to widen his view of the system state. This would lead to an in-
519 creased understanding of the relevant states as well as the possible
520 transitions between them. If the tester succeeds in this, false failures
521 due to running tests under the wrong system state may decrease.

523 4.1.2 **Risks. Generality:** Although the need to better understand
524 system states has been presented as a strength of the approach, this
525 was also perceived as an increased burden when writing the GAs.
526 This is due to the specification of the guard becoming much more
527 general: “You cannot assume anything [about the stimuli sequence],
528 so you need to specify when to test and when not to test.” [i2]

529 **Coverage:** Without direct control over the stimuli, there is a risk
530 that some GAs never get triggered and it is thus important to keep
531 track of which GAs have been covered. “For the stimuli part, it will
532 be difficult to know if I have covered all states i have to cover” [i6]

533 **False results:** The inherent generality of GA expressions contains
534 the risk of getting false results, e.g., due to missed signals in the
535 guard specification, or side effects from another function that was
536 overlooked by the tester. In summary, this could lead to a lot of
537 analysis whether a fault reported by an GA is a real fault or not.
538 “There is a risk that a lot of analysis is required to check whether
539 reported faults from the GA are real faults or not.” [i2]

540 4.1.3 **Applicability and Limitations.** In total, the applicability of
541 the approach was guessed by the interviewees to be within 40% to
542 80% of the test cases that they have today. In detail the following
543 categories were formed from the data:

544 **Integration level:** In general, the approach seems to be applicable on
545 different levels of integration, even though the answers varied. One
546 participant thought that, although technically possible, it would
547 not be motivating enough for the lowest test levels to adopt the
548 GA approach, while another considered it a perfect fit for unit tests.
549 On the ECU level, everything depended on the complexity of the
550 underlying functions and physical processes, but again not on the
551 approach per se. Finally, all interviewees agreed that the benefits
552 of the approach increase when climbing up the integration chain.
553 “The approach works better on a higher level with many ECUs” [i1].

554 **More applicable:** In the discussions, there were four kinds of tests
555 that were mentioned as particularly applicable for the approach.
556 These are *simple tests, non functional tests with long duration, posi-*
557 *tive testing, and ubiquitous requirements.* “The approach is good for
558 analyzing vehicle logs for simple faults that must never occur.” [i5]

559 **Type of faults:** There are also some types of faults that are believed
560 to be easier to find with the approach. One example is when signals
561 propagate between different ECUs and there are specific require-
562 ments on, e.g., timing. Another type of faults where the approach
563 seems particularly promising is glitches that occurs only occasion-
564 ally or faults that occur in unexpected parts of the system that you
565 are not currently focusing on.

566 **Less applicable:** Examples of where the approach is less applicable
567 are tests of diagnosis functions and fault injection. In such cases, the
568 fail states were perceived as difficult to model and the expressions
569 tend to be very lengthy. “1 out of 5 is fault injection related and not
570 suitable for the approach.” [i4] Tests with a lot of special cases or
571 exceptions also fall under this category. Another example is test
572 cases where the result cannot be automatically encoded, such as
573 testing visual output on the instrument cluster. The most discussed
574 barrier for using the approach is the complexity of the test case.

575 **Complexity:** Several different sources of complexity were identified
576 during the interviews, such as: number of inputs / signals, com-
577 plexity of the tested function / requirement, amount of intentional
578 functional interaction, number of exceptions / special cases, length
579 of state sequences, and number of possible state transitions.

4.2 Language

The collective results of this content area serves the purpose of answering RQ2. - *What are the potential benefits and drawbacks of using the T-EARS language to model GAs?* The analysis resulted in two categories directly contributing to the research question: Adoption and Ease-of-use. Aside from answering the research question, the analysis also revealed a number of suggestions on how to improve the language. This is presented in the category Suggestions. Lastly, we present a summary of a discussion with the participants on different, more specific, aspects of a test case written in T-EARS.

4.2.1 Adoption. This category describes the participants' experience when starting to use the language. Four of the participants were used to the way of thinking, since the format of their requirements were pretty similar, or were even written in the EARS format. For them, starting to use T-EARS was not a big step. At least not for the simple cases. When things got more complex, e.g., handling special cases, they experienced it as more difficult, mainly due to the lack of user defined functions. The remaining two participants had more difficulties than the others to start using the T-EARS language. The main reasons were problems with log compatibility to the SAGA tool and that they needed more example expressions to study. The overall impression of the language was that it is mature enough for simple expressions, but not yet mature enough for more complex tests. More examples of T-EARS expressions and visual support from the tool while experimenting, were mentioned as important things to quickly get up to speed with the language.

4.2.2 Ease of use. One of the most important aspects of T-EARS is how easy it is to use. Three sub categories were identified, representing factors that the participants thought contributed to or hindered ease of use: *Facilitators*, *Barriers* and *Complexity*.

Facilitators: The small set of keywords and the simplicity of the syntax were perceived to give a clear and easy to understand test code. *"... it is more clear how you reach a testable system state than our current Python code."* [i5] The resemblance with the requirement syntax, especially the case when the requirements are written in EARS was also appreciated. *"The requirements we have in the EARS format can probably be used directly in T-EARS."* [i3]

Barriers: The simplicity of the language also became a limiting factor, e.g., when expressing series of states, or long complex expressions. The semantic distinction between T-EARS keywords **while** (state) and **when** (event), especially in combination with timing, was a bit difficult to understand. *"I do not know how long "when" is valid. For example if I momentarily press a button to activate something, in reality it is still activated after the button has been released"* [i2].

Complexity: The sources of complexity in a T-EARS expression comes mainly from using the approach, see Section 4.1. *"The complexity lies in the required domain knowledge, signals, system states and transitions, rather than writing the test case."* [i2] Nevertheless, a few ideas for reducing the complexity are presented below.

4.2.3 Suggestions. The last category was the participants suggestions on how to make the language easier to use. The main theme here was the ability to deal with more complex tests.

User-defined functions: The most discussed feature was the ability to create user defined functions. Either for grouping or encapsulate long expressions or extend the available convenience functions, e.g. for simplified signal processing.

Higher-level state variables: This is a variant of the user defined function that was discussed. In practice, this means that it would be possible to use information on another GA (guard activated, passed, fail) as input to another GA.

GA-scoping: Scripted activation and deactivation of groups of GAs was suggested, using different mechanisms: *i)* Triggered if another GA has been triggered (one usage of higher level state variables as discussed above); *ii)* Implementing the **where** keyword from EARS. This would allow the inclusion of GAs, relevant to a specific configuration only: *"...there are certain functions that do not exist in all vehicles, and those you need to consider..."* [i2]

Save / restore signal values: The last suggestion was to include a save / load mechanism, e.g., to be able to store a signal value when a certain guard is activated. The stored signal should be possible to use as an input to another GA. This is sometimes useful in more complex functions, such as cruise control, where a signal needs to maintain its value during a period of time.

4.2.4 Detailed impressions. The following is a summary of the discussions regarding specific aspects of the T-EARS language. The interviewees were asked to compare T-EARS with their current test specification language, to see if using T-EARS for specification increases or decreases, e.g., complexity.

Ambiguity: Test cases written in T-EARS would be less ambiguous. *"T-EARS can be used for avoiding misunderstandings."* [i3] If the requirements are written in EARS, it will be less ambiguous since virtually no translation is required.

Correctness: Same or better. *"Since everybody will be forced to write the same way, it will be harder to do it wrong. At least for short and simple expressions."* [i6]

Understandability: At the same level or better for not too complex tests. If the requirements are expressed in EARS, it will be substantially easier for a broader range of people to understand the resulting test case, especially for other testers and non-programmers. In the case of more complex test cases, one of the participants believed that the current language (Python) would result in a test that is easier to understand.

Consistency: Whether the usage of T-EARS would give more or less consistent test cases than using, e.g., Python, differed depending on what the participants included in the language. If considering only the T-EARS core language, and not support functions, the answer was that the tests would be more consistent, since the T-EARS code would look the same for all test cases. Some participants had higher level functions in their test frameworks, effectively standardizing the test code to be consistent. Finally, the time specification feature of T-EARS was emphasized as an important factor: *"...these timing expressions, we think this is the whole difference and it really helps us in getting the test cases consistent."* [i3]

Conciseness: Conciseness depends on the system under test and ultimately the complexity of the requirement. For simple requirements it should be more concise, but for more complex it might get less concise, since it is not evident how to encapsulate expressions into functions, as in, e.g., Python.

Design / Level Independence: The participants did not see any technical limitations for using the language at different test levels.

Stability: The choice of specification language is not considered to affect stability, at least in any dominant way. However, for some groups, being able to specify timing in the tests were a synonym

581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696

for stability. Even though some of the interviewees had this feature in their testing framework, the participants appreciated the built-in temporal aspects of T-EARS.

Complexity: The complexity of the test cases written in T-EARS depends to a great deal on the complexity of the tested functions. For simple functions the resulting test cases are less complex, but due to the simplicity of the language, e.g., lacking user function or other encapsulation mechanisms, the complexity is expected to be greater when testing more complex functions.

4.3 Tool

The results of this content area, divided into categories *Potential* and *Challenges*, serve the purpose of answering RQ3. - *What are the potential benefits and requirements on an interactive tool for the development of GAs?* In addition, suggestions from the case study participants on how a tool should meet the challenges are included.

4.3.1 Potential. Interaction: The strength of the plots in the graphical signal view is that the relevant signals, as well as the intermediate evaluation results, are shown as the GA expression is being typed in, giving an *immediate feedback* on the current expression to the tester. *“It becomes pretty obvious when certain signals will be evaluated”* [i2].

Tool integration: The participants found a great potential in writing GA expressions, not only due to the T-EARS language, but also due to the integration between the GA editor, recorded signal logs, and the graphical signal and test evaluation view.

Effectiveness: The possibility to load another persons log while fine-tuning the test case and immediately get the graphical feedback of the result, was perceived as a great help for increasing the *confidence in the correctness* of the test case, as well as a great tool for *fault finding*. *“It is super useful to directly see where it went wrong, and HOW wrong it went, for example timing problems or that a signal never reaches a value.”* [i4]

Efficiency: *Off-line analysis of log data* also saves a substantial amount of rig execution time, since tests do not require re-execution on the rig, but can rather be analyzed on a personal computer. *“You can run it on your own machine and get direct response on what you write. Instead of waiting a week to get rig time.”* [i5]

4.3.2 Challenges. Increased-data-complexity: As discussed in Section 4.1.1, the approach may lead to an increased number of test cases in the form of GAs. To reach the full potential of the approach, it should be possible to apply a GA on several log files: *“You would like to run a GA on many, maybe 1000 different log files to see that it really works.”* [i2] It must also be possible to apply several GAs on a single log file: *“It would be good to be able to run all logs to see if the fault repeats.”* [i1] This poses some requirement on the test management system and requirement handling tools: *“The whole chain should be automated, e.g analysis of data when it is produced, creating reports and logging into results data base.”* [i5]

Complex cases: In general the interviewees considered the tool to be rather easy to use for testing simple functions. Due to a certain immaturity of the tool, more complex test cases were more challenging. One example is that long expressions resulted in many plots. *“These [the tested GAs] were pretty short GAs, still, it required a lot of scrolling to see relevant signal states in the plots.”* [i2]

Missing signals: The testers that encountered problems due to log data incompatibility or missing signals in the log file, experienced

more difficulties in expressing the guards. *“Without recorded data, it may be difficult to understand the state you need to be in.”* [i2] This emphasizes the importance of supporting a wide range of log-formats e.g, CANalyzer and VISION log formats.

Huge signal spaces: Further, in order to make the expressions more readable, short names are shown in the current version of the editor. In systems that have the same short-names in different contexts, e.g. different CAN buses, it can be hard for the tester to tell which one is used in an expression. Also difficulties finding the signals in the current signal list was mentioned. *“Many signals have the same [short] name in different contexts, that is a problem.”* [i2] Filtering, sorting and grouping of signals were discussed as solutions to the challenge. Other suggestions were drag and drop support from the signal list and improved auto-completion of the complete signal paths. Further, means of evaluating temporary expressions, not part of the current GA, were requested: *“Some times, when something has gone wrong, you want to inspect variables that is not part of the [GA] expression.”* [i5]

Visualization: While useful for one tester, another tester may find the number of plots a bit daunting, especially for a long expression, and if the signals are plotted in separate graphs. *“It was a bit difficult to understand the many graphs that showed up.”* [i2] Another challenge was the sample points, that some times are needed, but sometimes are overloading the plots. The challenge of visualizing missing cyclic signals and GA coverage was also discussed.

Navigation: In some cases, interesting sections in a plot are spread out in time and difficult to find, e.g., spikes in a signal which are hard to see if not zoomed in. In response to such navigation challenges, different search features of the plot were suggested, e.g., go to next spike and moving the timeline while in zoom mode.

Performance: Since the current log files are mainly ASCII, the testers suspected performance problems as the log files grow large, since the startup time and responsiveness of the tool degraded for very large log files.

Realtime: Several interviewees expressed a wish to be able to run the tool in the same way, performing the same analysis, while providing input data in realtime. This requires that the toolbox observes and keeps track of the SUT's current state, which is an additional challenge, not least due to the risk of such an observer growing exponentially.

5 RELATED WORK

Brzeziński discusses the distinction between active and passive testing in [2] and concludes that the usual meaning of the word *testing* is active testing, which also means that testers are normally unfamiliar with passive testing. We try to address the discrepancy of the mental model of testing and the notion of passive testing with 1) the interactive round-trip in the toolbox, and 2) the T-EARS language. The case study shows that these can successfully enable development of passive tests, which means that such things as parallel execution of tests (which is easy using a passive testing approach) will be more accessible to testers.

EARS has been available since 2009 [12], and it has since been used more and more in industry [11, 13]. It has also been formalized for, e.g., deriving use cases from requirements [10]. We have instead extended EARS, into the T-EARS language, to be an efficient way

to describe passive tests in the GA approach. Our case study shows that T-EARS is perceived as a good way to represent and reason about passive tests up to a certain complexity level. Above this complexity level the interviewees think the T-EARS expressions will be too complex to write and read.

Runtime verification is the monitoring of a system's execution and checking it against properties described in some logic [9]. We have not investigated whether T-EARS is as expressive as e.g. temporal logic. Instead, our intention with the interactive work flow has been to provide testers with a means to develop concise and correct tests without needing to know any temporal logics.

Property-based testing, first popularized with the Haskell QuickCheck tool [3], aims at capturing properties of a program and then testing this property on a large number of inputs. A found counterexample is displayed to the user. QuickCheck is built around the possibility to inject a large number of test stimuli that test the properties. The studied toolbox, however, aims at developing tests without the need to have access to the SUT.

There are examples of usages of passive testing and runtime verification in the automotive domain [14, 18], as well as commercial tools like EmbeddedSpecifier and Montimage Monitoring Tool (MMT). To the best of our knowledge, these examples and tools are not using an interactive tool for test/monitor development, nor has it been scientifically evaluated in the context of our research questions. Thus, there is no documented knowledge how easy/hard users find it to express passive tests and monitors in these tools. Our findings are that an interactive work flow and a well chosen language can make passive testing approachable.

6 CONCLUSIONS AND FUTURE WORK

The aim behind the presented work was to assess whether passive testing realized using the GA approach, its tool realization, as well as the T-EARS language could be used advantageously in the industrial practice. And conversely, what is missing in order for the method and the tool chain to take off. For this sake, an exploratory case study was performed, based on semi-structured interviews with experienced test engineers within the automotive domain.

The case study confirmed that it is rather easy to understand the approach, learn the T-EARS syntax, and start using the tool. Once the learning threshold is overcome, the idea behind passive testing, and particularly the GA approach, seems quite attractive to the practitioners, resulting in tests that read almost as executable requirements. The approach was perceived to result in more clear, realistic, and robust tests, at least when the test complexity was not too high. This was estimated to hold in 40-80% of legacy test cases.

On the downside, most interviewees agreed that, due to limitations of the T-EARS language, more complex tests could lead to unmanageable logical expressions. Also, in order to write a good GA, general enough to handle any kind of test input correctly, it is important to understand precisely which parts of the system that can affect the testable state. Given that test stimuli are not controlled, concerns were raised about the requirement coverage and the diversity of inputs. Various log formats should be supported.

Ideas about the future work flow naturally from the conclusions of this case study. Besides direct actions on improving the tool in response to the comments from the interviewees, there is a need to look into more challenging questions. One such is how to manage

rising test logic complexity. Would it be possible to adapt T-EARS to manage more complex cases, e.g. through encapsulation of T-EARS logic into more complex expressions, or would the user-friendliness of the approach get lost? Besides supporting different input formats, it would be interesting to consider the generation of test stimuli in more depth. Ideally, it should be possible to choose some criteria, e.g. requirement coverage, functional interference, or input diversity, and generate realistic inputs automatically. It is also important to be able to scale with the rising amount of GA logic, input data, as well as auxiliary information, both when it comes to responsiveness and visualisation properties of the tool. Finally, it will be important to detect non-feasible inconsistencies, either in GAs or in generated stimuli, that lead to incorrect results. Ideally, this should be done algorithmically, and learned so that similar mistakes are avoided.

REFERENCES

- [1] César Andrés, Mercedes G Merayo, and Manuel Núñez. 2008. Passive Testing of Timed Systems. In *ATVA*. Springer, 418–427.
- [2] Krzysztof M Brzeziński. 2011. Active-passive: On Preconceptions of Testing. *Journal of Telecommunications and Information Technology* (2011), 63 – 73.
- [3] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*. ACM, New York, NY, USA, 268–279. <https://doi.org/10.1145/351240.351266>
- [4] Daniel Flemström, Thomas Gustafsson, and Avenir Kobetski. 2017. SAGA Toolbox: Interactive Testing of Guarded Assertions. In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*. IEEE, 516–523.
- [5] Ulla H Graneheim and Berit Lundman. 2004. Qualitative content analysis in nursing research: concepts, procedures and measures to achieve trustworthiness. *Nurse education today* 24, 2 (2004), 105–112.
- [6] Sarah Gregory. 2011. Easy EARS: Rapid application of the easy approach to requirements syntax. In *19th IEEE International Requirements Engineering Conference (RE'11)*.
- [7] T. Gustafsson, M. Skoglund, A. Kobetski, and D. Sundmark. 2015. Automotive system testing by independent guarded assertions. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW'15)*, 1–7.
- [8] Elizabeth Hull, Ken Jackson, and Jeremy Dick. 2010. *Requirements engineering*. Springer Science & Business Media.
- [9] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming* (2009), 293–303.
- [10] Dipankar Majumdar, Sabnam Sengupta, Ananya Kanjilal, and Swapan Bhat-tacharya. 2011. *Adv-EARS: A Formal Requirements Syntax for Derivation of Use Case Models*. Springer Berlin Heidelberg, Berlin, Heidelberg, 40–48.
- [11] A. Mavin and P. Wilkinson. 2010. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In *2010 18th IEEE International Requirements Engineering Conference*. 277–282. <https://doi.org/10.1109/RE.2010.39>
- [12] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. 2009. Easy approach to requirements syntax (EARS). In *17th IEEE International Requirements Engineering Conference, RE'09*. IEEE, 317–322.
- [13] A. Mavin, P. Wilkinson, S. Gregory, and E. Uusitalo. 2016. Listens Learned (8 Lessons Learned Applying EARS). In *2016 IEEE 24th International Requirements Engineering Conference (RE)*. 276–282. <https://doi.org/10.1109/RE.2016.38>
- [14] P. Mouttappa, S. Maag, and A. Cavalli. 2013. Monitoring Based on IOSTS for Testing Functional and Security Properties: Application to an Automotive Case Study. In *2013 IEEE 37th Annual Computer Software and Applications Conference*. 1–10. <https://doi.org/10.1109/COMPSAC.2013.5>
- [15] Amalinda Post, Igor Menzel, Jochen Hoenicke, and Andreas Podelski. 2012. Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH. *Requirements Engineering* 17, 1 (2012), 19–33.
- [16] Guillermo Rodriguez-Navas, Avenir Kobetski, Daniel Sundmark, and Thomas Gustafsson. 2015. Offline Analysis of Independent Guarded Assertions in Automotive Integration Testing. In *The 12th IEEE International Conference on Embedded Software and Systems (ICSS)*.
- [17] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131–164.
- [18] Konstantin Selyunin, Thang Nguyen, Ezio Bartocci, and Radu Grosu. 2016. *Applying Runtime Monitoring for Automotive Electronic Development*. Springer International Publishing, Cham, 462–469. https://doi.org/10.1007/978-3-319-46982-9_30

813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870

871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928