DOI: 10.15439/2023F6171
ISSN 2300-5963 ACSIS, Vol. 35

Risk-Based Continuous Quality Control for Software in Legal Metrology

Marko Esche, Levin Ho, Martin Nischwitz, Reinhard Meyer
Physikalisch-Technische Bundesanstalt,
Abbestraße 2-12, 10587 Berlin, Germany
Email: marko.esche@ptb.de, levin.ho@ptb.de, martin.nischwitz@ptb.de, reinhard.meyer@ptb.de

***Abstract*—Measuring instruments are increasingly defined by complex software while using simple hardware sensors. For such systems, software conformity between certified prototypes and devices in the field is usually demonstrated using version numbers and hashes over executable code. Legal requirements for regulated instruments could equally be satisfied if prototype and device in the field display identical functional behavior even if hashes differ. Such functional identification can give instrument manufacturers room for software patches and bugfixes without the need for recertification. Based on the L^* algorithm, which is used to learn the language which deterministic finite automata accept, a risk-based method is proposed that realizes automatic functional identification of software to a certain extent, thereby enabling quality control of regularly updated measuring instruments without the need for frequent manual inspections. Risk assessment may be used to identify critical state transitions in monitored devices, which can be used to trigger recertifications if needed.**

I. Introduction

MODERN communication infrastructure and the ubiquitous availability of significant computation power even in small devices like smart watches and smart sensors allow software developers to remotely and regularly fix bugs identified during use of an IT component in the field. The same mechanism can also be used to deliver upgraded software with new features to remote devices, enabling IT equipment manufacturers to sell devices based on long-living hardware which can be upgraded to customers' needs through software updates. However, this development also comes at a certain cost: Remote update capabilities have proven to introduce unexpected or unintended errors into otherwise stable systems [1]. Therefore, approaches to cover this gap (without forcing potential users of updated software to validate the complete source code of a device) have received significant attention in recent years [2], [3]. Such approaches enable users of IT equipment to monitor a device's behavior for potential anomalies without having to check each individual update, thus providing a high-level approach to identify software by means of its functionality rather than by means of its bit pattern.

Monitoring and identifying a device's functional behaviour becomes especially important if requirements on these devices are mandated by legal regulations, typically involving recertification of the entire system in case of modifications. One industry sector affected by such regulations is Legal Metrology covering all measurements conducted in the European Union for commercial or official use. These regulated instruments include, e.g., taximeters for taxi fare calculation, gas meters for measuring gas consumption and length measuring instruments to determine the dimensions of sold goods. Any such instrument put on the common EU market has to be subjected to a conformity assessment procedure according to Annex II of the Measuring Instruments Directive 2014/32/EU (MID) [4]. One conformity assessment body performing this task is Germany's national metrology institute Physikalisch-Technische Bundesanstalt (PTB).

During conformity assessment, manufacturers have to demonstrate that their instrument fulfills the essential requirements given in Annex I of the MID. During use, market surveillance authorities across the EU monitor devices and their usage to detect potential non-compliance. As an example, essential requirement 8.3 states that, ”Software that is critical for metrological characteristics shall be identified as such and shall be secured. Software identification shall be easily provided by the measuring instrument. Evidence of an intervention shall be available for a reasonable period of time.“ [4]. Not only does this entail the identifiability of software in general, but also the possibility to detect changes to said software and make them evident to all parties involved. Typically, both identifiability and detection of changes are achieved by using cryptographic hashes over the executable code to identify specific versions of the software and to detect unwanted modifications [5]. However, such an approach quickly may put serious strain on conformity assessment bodies and market surveillance authorities alike. For example, even a recompilation of otherwise unchanged source code my result in a different hash due to the inclusion of compile time stamps etc. Therefore, solutions that automatically evaluate software modifications and link them to a potential risk of non-compliance are needed.

To this end, a novel risk-focused method for remotely monitoring software in devices subject to legal control is proposed here. It is envisioned that the method will be used by market surveillance authorities and inspectors to automatically check certified devices in the field for potential non-compliant behavior. If a device is deemed to be in violation of legal requirements after a software modification, the manufacturer would then be requested to resubmit the modified software for a complete conformity assessment procedure. The main contributions of the paper are the following: The proposed method constitutes a first step towards automatic remote quality control of devices subject to legal control. It enables automatic selection of risk scenarios based on remotely obtained behavioral data and thus also realizes functional identification of software to a certain extent.

The remainder of the paper is structured as follows: Section II provides some background on modelling and learning algorithms and presents the current state of the art in quality control for software as well as the risk assessment method currently used in Legal Metrology in the European Union. In Section III, the concept of modelling certain types of measuring instruments as deterministic finite automata (DFA) is investigated. The section also outlines which preconditions need to be fulfilled to justify such an approach. Afterwards, Section IV describes a novel risk-focused method of monitoring evolving software in measuring instruments based on the Active Continuous Quality Control (ACQC) approach from [2]. The method is then experimentally tested and evaluated in Section V. Finally, Section VI summarizes the paper and provides suggestions regarding further work.

II. Background and Related Work

Certain types of algorithms can be described as finite automata. The corresponding models and how to learn the behavior of such algorithms, which is of particular interest during monitoring of potentially modified software, are described in Section II-A. The methods proposed by Neubauer, Windmüller and Steffen together with Howar and Bauer [2], [3], which apply active automata learning to quality control for evolving systems, are briefly described in Sections II-B and II-C before recapitulating the previously published method of risk assessment for measuring instruments in Legal Metrology in Section II-D.

A. Active Automata learning

Simple state machines steered by input symbols from a finite alphabet that trigger internal state changes can be used to describe the behavior of certain types of algorithms such as used in controllers for elevators, household appliances, simple digital watches etc. [6]. From a mathematical point of view, these state machines, also referred to as DFAs, are defined as a 5-tuple $(Q, \Sigma, \delta, q_0, K)$ [6] where:

1) Q is a finite nonempty set of states.
2) Σ is a finite input alphabet.
3) $\delta : Q \times \Sigma \to Q$ is the transition function. (1)
4) $q_0 \in Q$ is the initial state.
5) $K \subset Q$ is the subset of accept states.

To indicate whether an arbitraty input sequence has successfully been processed, DFAs may contain accept states K which then trigger an *accept* message if such a state is reached. Otherwise, the output would be a *reject* message. It should be noted that the set K may also be empty, implying that the DFA does not contain any accept states triggering an *accept* message. In practice, the output of an algorithm is usually more complex than such binary feedback, requiring the existence of an output symbol from a finite output alphabet Γ. Such more general state machines are referred to as Mealy automata, which can be characterized as a 6-tuple $(Q, \Sigma, \Gamma, \delta, \gamma, q_0)$ [7] where:

1) Q is a finite nonempty set of states.
2) Σ is a finite input alphabet.
3) Γ is a finite output alphabet.
4) $\delta : Q \times \Sigma \to Q$ is the transition function. (2)
5) $\gamma : Q \times \Sigma \to \Gamma$ is the output function.
6) $q_0 \in Q$ is the initial state.

In addition to the transition function δ, describing state changes depending on the input symbol, these also possess an output function γ that associates an output symbol with each state change. Such Mealy automata were originally conceived to represent arbitrary logic circuits and can even mimic complex IT systems at a certain abstraction level [2]. For additional details, see Section II-B. To infer a DFA without having to know the exact implementation, the L^* algorithm was developed by Dana Angluin in 1987 [8]. The algorithm was later extended to the L^*_M algorithm to learn properties of the more general Mealy machines as well. Given that software changes in measuring instruments may have unknown effects and the instrument itself thus takes on the characteristics of a system with unknown behavior after an update, the basics of the L^* shall be briefly summarized here. See the original publication by Dana Angluin [8] for additional details of the L^* algorithm and the paper by Shahbaz and Groz [9] for an extended discussion including the L^*_M extension:

The aim of the L^* algorithm is to determine the properties of an unknown DFA by means of so-called membership and equivalence queries. To this end, the L^* learner communicates with a teacher T. The teacher abstracts the system under test (SUT), so that generic queries may be used by the learner to determine the SUT's internal DFA. If $L(A)$ is the set of strings a SUT A accepts, i.e., its language, and $Aut\,(A)$ is the set of all finite state machines with input alphabet Σ then the two types of generic queries used by the learner can be defined as follows:

- Membership queries $Q_M : \Sigma^* \to \{0, 1\}$ where the learner asks the teacher to test the SUT with a given string x from the free monoid Σ^* that contains all words over Σ. If $x \in L(A)$ the response of the teacher is 1, otherwise 0.
- Equivalence queries $Q_E : Aut\,(\Sigma) \to \Sigma^* \cup \{\text{true}\}$ where the learner L^* asks the teacher T to perform an equivalence test between the current learned automaton representation $A' \in Aut(\Sigma)$ and the SUT A, resulting either in a counterexample $c \in \Sigma^*$ or confirmation of the equivalence.

Internally, the L^* algorithm operates on a so-called observation table that stores results of the queries in a systematic fashion. To this end, the learner continually performs membership queries until it has constructed an initial model A'. Subsequently, it issues an equivalence query to the teacher, which either confirms correspondence between A' and A or responds with a counterexample $c \in \Sigma^*$ that fulfills either $c \in L(A) \wedge c \notin L(A')$ or $c \notin L(A) \wedge c \in L(A')$. The algorithm finishes if the obtained data is sufficient to generate a system with the same algorithmic behavior as the SUT A. To illustrate the outcome of the L^* algorithm, an exemplary DFA is described in Section III together with a resulting transition function δ in tabular form in Table I.

B. Active Continuous Quality Control (ACQC)

In [2] Windmüller, Neubauer, Steffen, Howar and Bauer presented a novel approach for ensuring compliance of evolving complex applications through active automata learning technology. Their goal is to supervise and control modifications of applications during their entire life cycle. This is realized by establishing a consistent level for comparison via adaptive behavioral abstraction. Abstraction is achieved by means of a user-centric communication alphabet, where elements of the alphabet may correspond to entire (complex) use cases. One advantage of the method lies in its capability to identify bugs by simple examination of so-called "difference views“ between consecutive models. The authors observe that software testing in general is not tailored to keep up with current, continuously evolving component-based software systems since repeatedly updating test suites for such systems is time-consuming and expensive. In [2] incremental active automata learning technology (also referred to as test-based modeling) is employed to address this issue.

To this end, daily system builds with an integrated fully automatic testing process are used, where the testing process is controlled by incremental active automata learning. The proposed approach aims to address the following main problems:

- "Stable abstraction“: Downward compatibility is assumed, meaning users of the system should not change the way they interact with the system. Nevertheless, the source code etc. may change, but such modifications should not be apparent to the user. Therefore, the chosen abstraction mechanism is oriented on the level of use cases to facilitate comparisons between different software versions. Subsequently, the user-centric communication alphabet reflects distinct activities as part of the use cases.
- "Bridging implementation“: A mechanism of the common abstraction level must ensure that any test is supported by a correct (version-dependent) implementation of an adapter for the symbols of the alphabet.
- "Maximal reuse“: The central aspect of ACQC is based on the L_M^* learning algorithm for model inference. Based on selected counterexamples, the algorithm infers models from executed tests, see Section II-A. One drawback of the approach is the computationally expensive tests needed for the active learning process.

The authors observe that hypothesis models for new software releases are derived at the same level of detailedness as for the previous software release, which constitutes the main advantage of ACQC over similar approaches. Since identification of counterexamples is inherently ineffective, the derived system description will improve over time. Obviously, a precise initial model is needed to enable model-based testing. According to Windmüller, Neubauer, Steffen, Howar and Bauer, derivation of such models from source code is impractical for systems of a certain size. Indeed, any form of use-case-level modelling is difficult for such systems. Instead, active automata learning is used to extract models from live systems. The learned models then serve as the basis for regression tests. This approach will be reused in the method to be investigated here, see Section IV.

In [2] the proposed continuous quality control approach was validated by applying it to the Online Conference Service used for submitting and reviewing publications at Springer Verlag as an example with specific use cases as input symbols. Correspondingly, each input symbol represents processes like paper submission, reviewer selection or review submission. With such a high-level representation, a reasonably stable abstraction (as required above) was realized. The authors found that the chosen high-level modelling of input and output alphabets as abstraction of different use cases is well suited as a quality management facility for evolving IT systems. Not only is their method able to detect bugs, it also verifies if functional behavior of a system remains unchanged from one release to the next.

It should be noted, however, that the model learned by the L_M^* algorithm does not directly provide a link between the known set of states Q and the derived transition function δ. Instead, most L^* and L_M^* implementations assign input symbol sequences to the states they lead to. If the binary input 0 leads from a transition from the default empty state $\{\}$ to a state A, that state will be represented by the input sequence 0. If another input symbol 0 then leads to a transition from A to B, whereas the alternate input symbol 1 leads from A to C, B would be represented as 00 and C as 01. From a theoretical point of view, this corresponds to building the equivalence classes of the automata congruence relation for all states. It follows that an outside examiner can match the learned transition function δ against a known reference, but it is not guaranteed that the mapping between known states Q and learned states Q' is correct. This observation will be revisited again and illustrated by a more detailed example in Section V.

C. Risk-Based Testing via Active Continuous Quality Control

In [3] Neubauer, Windmüller and Steffen extended their approach to active automata learning and testing by adding a risk prioritization component. In this context, risk assessment is used to produce alphabet models which help to control the ACQC process to increase coverage of risk scenarios. The authors explain, that today's complex IT systems usually consist of a combination of application servers with webinterfaces and third-party services. Due to the resulting heterogeneous structure, the subsequent system behavior becomes increasingly difficult to predict. During updates in particular, the mix of modified functionality and upgraded third-party components may have unintended effects. Their aim, therefore, was to continually perform automatic quality checks while using risk assessment to reduce the manual labor involved in regression testing.

In this regard, platform migrations are of particular interest since user experience may drastically change, even though the underlying functionality was not intended to be modified. Of course, potential risks resulting either from a platform change or from modified functionality cannot be automatically inferred. Therefore, the authors amended the original ACQC approach from [2] by enabling risk analysts to identify critical system aspects and prioritize them for error detection during the automatic model inference and checking steps. However, the paper [3] does not specify how risk levels are formally determined. Since risk analysts are typically not involved in software development itself, it becomes necessary to provide them with an abstraction layer that can be included in the original ACQC approach without performance loss. To this end, Neubauer, Windmüller and Steffen used the already abstract alphabet symbols from [2], which model different use cases of the SUT (see Section II-B).

The authors of [3] acknowledge that there are also model-driven approaches to risk-based testing such as the one described by Lund, Solhaug and Ketil Stølen in [10]. The so-called CORAS methodology offers the possibility to perform risk assessment using well-defined software models based on UML and the Unified Process (UP). However, CORAS and similar approaches only address the modelling aspect for risk assessment and are unable to monitor and perform comparisons between subsequent models of SUTs. Neubauer, Windmüller und Steffen also observe that it is unrealistic to assume that the internal number of states of a system will not change during its lifecycle. They therefore propose to continually repeat the learning process. This will be mirrored in the approach presented here, see Section IV.

D. Software Risk Assessment in Legal Metrology

One mandatory element of conformity assessment for measuring instruments consists of carrying out and evaluating a risk assessment for the instrument or type pattern to be assessed. In [11] Esche, Grasso Toro and Thiel described a method for software risk analysis particularly tailored for the software of such systems. The method is based on ISO 27005 [12] and ISO 18045 [13] and makes use of so-called assets, e.g., software, measurement data and parameters, and matching security properties, i.e., integrity, authenticity and availability, derived from the essential requirements from Annex I of the MID. These assets include the software, parameters and data of the instrument, but also the indication of the result, accompanying inscriptions and stored data. Within the frame of this paper, only the data during processing shall be considered. For such data, the MID requires integrity, authenticity and availability, i.e., it must be ensured that data cannot be modified or deleted without detection and that they are traceable to a known source. In short, any inadmissible influence on the processed data must be detectable.

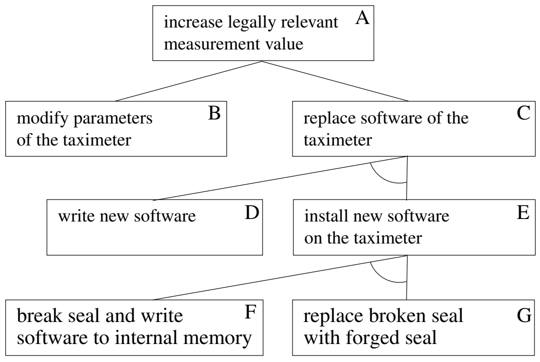
increase legally relevant measurement value
A
modify parameters of the taximeter
B
replace software of the taximeter
C
write new software
D
install new software on the taximeter
E
break seal and write software to internal memory
F
replace broken seal with forged seal
G

Fig. 1. Graphical representation of an attack tree that illustrates necessary steps to manipulate the calculated fare of a taximeter during processing [14]. Child nodes must be read as OR-connected, unless they are connected by an arc, which represents an AND-connection [11].

The first step of a risk assessment then consists of formulating certain threats that constitute an invalidation of any security property for the assets. For example, such a formal threat might read, "An attacker manages to invalidate integrity or authenticity of measurement data during processing.“ Given the known properties of the instrument, the assessor then identifies potential attack vectors which encompass practical technical steps to be implemented to realize the threat. Since such attacks tend to be made up of several steps which may even be shared between different threats, Esche, Grasso Toro and Thiel introduced the concept of Attack Probability Trees (AtPT) in [11]. An AtPT can be used to divide complex attacks into smaller subgoals by means of a tree representation, see Figure 1 for an example addressing attacks on the calculated fare of a taximeter. The AtPT method may be seen as an example of fault tree analysis (FTA) with an added layer that links the method to the vulnerability analysis from ISO 18045 enabling users of AtPTs to rank threats by means of a formalized and well-defined risk assessment. In the Figure, node A, which corrsponds to the threat of manipulating the measurement value, is divided into nodes B and C which represent the alternatives of either manipulating the measurement parameters or replacing the software of the instrument. These two child nodes may be split into further subtrees themselves. Once the tree has been established, all leaf nodes are assigned scores for required time, needed expertise, knowledge of the system, window of opportunity for an attacker and necessary equipment in accordance with the corresponding guidelines from ISO 18045 [13]. Finally these scores are propagated up the tree as prescribed by the rules from [11] to calculate probability of occurrence score and impact score of the original threat represented by the root node. The product of both, rounded to the next integer number, then becomes the (ideally) reproducible, numerical representation of the risk associated with the threat.

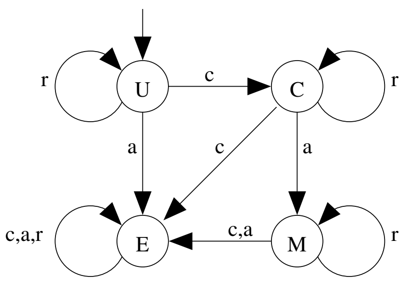

Fig. 2. DFA representing the different states of a heat meter and the state transitions. The heat meter states are: U for an unconfigured device, C for a configured yet inactive device, M for a measuring device and E for a an error state. The input alphabet consists of the symbol c for a configuration dataset, a for an activation signal and r for a request to retrieve measurement data from the device.

III. Modelling Measuring Systems as Deterministic Finite Automata

In principle, finding mathematical representations, i.e., functional identifications, even for simple measuring instruments is a complex task since various physical influences need to be taken into account and must be reflected in a corresponding uncertainty budget [15]. Automatic detection of unwanted behavior of complete instruments thus quickly becomes unfeasible. Nevertheless, many commonly used instruments, e.g., heat meters, typically contain internal state machines which ensure that the instrument behaves differently during installation/configuration than during permanent use. Among other qualities, heat meters have to guarantee that the installation point (either on supply side or return side of a heat generating device) can only be set during configuration and that the state cannot be reached again without physically tampering with the device. From this example, it should be clear that state machines within such instruments share many properties with DFAs and can thus be used to provide a simple form of high-level functional identification.

In the simple heat meter example, $Q = \{U, C, M, E\}$ would consist of the states U for an unconfigured device, C for a configured yet inactive device, M for a currently measuring device accumulating the consumed energy into a register and E for a device in an error state. A simple input alphabet would consist of three symbols $\Sigma = \{c, a, r\}$ where c represents a configuration datagram, a is the activation signal and r is a request to read measurement data from the device. For illustration purposes, a graphical representation of the complete DFA is given in Figure 2. It should be noted that the depicted DFA is only a simplistic exemplary representation of the possible software states of a heat meter. A real device will likely contain more states and more possible transitions. Also, the shown DFA only addresses the software aspects of the meter. For instance, if the permanent error state E is reached, recovery might still be possible via a hardware reset which is beyond the representation capabilities of the selected model. The corresponding transition function δ, which maps a current state to the next state given a specific input symbol, is shown in Table I. There exist some approaches to

TABLE I
Transition function δ for the heat meter example in Fig 2. State U represents an unconfigured device, C a configured yet inactive device, M a measuring device and E a device in an error state. c is the input symbol for a configuration dataset, a for an activation signal and r for a request to retrieve measurement data from the device.

		symbol		
		c	**a**	**r**
state	**U**	C	E	U
	C	E	M	C
	M	E	E	M
	E	E	E	E

model both, the measuring function and the DFA of measuring instruments, resulting in a so-called digital twin describing an instruments behavior under arbitrary conditions. However, these are not suitable to monitor and evaluate frequent software changes. As stated in Section I, mechanisms are needed that can automatically identify and evaluate software modifications. It should be clear from the above-mentioned example for heat meters, that some measuring instruments contain state machines that control the interpretation of sensor data to produce the measured quantity value.

While certain measuring instruments include internal DFAs controlled by external input [5], such instruments usually also produce variable output data - namely the measurement result - either in a digital or visual representation. Therefore, such systems fulfill the criteria of the more general Mealy automata. Nevertheless, as illustrated by the heat meter example above, many measuring instruments already contain simple DFAs enabling the use of the original L^* algorithm without having to define additional output alphabets and resorting to the correspondingly more complex L^*_M algorithm for Mealy automata. This approach also mirrors the fact that evaluation of software security aspects in measuring instruments and evaluation of the measurement functionality are usually two separate tasks during conformity assessment of such devices. Section IV will revisit this aspect when elaborating on a possible quality control strategy for measuring instruments in the field.

IV. Risk-Based Continuous Quality Control for Measuring Systems

In [3], the authors used risk assessment to prioritize the input alphabet for the L^*_M algorithm applied to a Mealy machine to ensure quick detection of potential implementation or migration errors in evolving IT systems. In the scenario where software is updated in measuring instruments subject to legal control, a little more flexibility might be possible given that mere bugfixes, which do not affect the functionality of the instrument, should be covered by the original conformity assessment certificate without the need to revise the certificate. To achieve this, the focus shall not be put on the choice of the input alphabet but rather on the state transitions δ discovered by executing the L^* algorithm for a new or unknown system. A graphical representation of the automatic quality control method proposed here may be found in Figure 3.

As discussed in Section II-D, performing and evaluating a software risk assessment has become an integral part of conformity assessment for most measuring instruments in the EU. During such an assessment, the risks assigned to individual threats or their subgoals can be used to derive a list of critical state transitions that the evaluator deems to be in violation or facilitate violation of the essential requirements from the MID, see top-left corner of Figure 3. If necessary, the numerical risk scores for individual threats described in Section II-D could be used to rank new state transitions according to their risk level. In the heat meter example from Section III, one such critical transition would be reverting from measurement state M back to the configuration state C, potentially leading to modified measurement parameters while a device is in use. The conformity assessment procedure could also be used to perform an initial execution of the L^* algorithm in a known environment. The initially discovered transition function δ and the known remaining elements of the DFA shall together be referred to as the model M_{old}. Continuous repeated learning of the DFA (right-hand side of Figure 3) will produce potentially modified models M_{new} which can be compared against the previously learned and accepted model taking into account the identified list of critical state transitions. As long as no critical transition is identified, the learning loop could be repeated indefinitely to ensure that the system still operates within certified functional limits. The updated model representation also allows human evaluators to graphically identify the recent software changes and determine their potential effect. Of course, model comparison only allows inspection of the internal DFAs of measuring instruments, neglecting to address the measurement function itself. However, this approach is also used in many conformity assessment bodies in the EU where software examination (focussed on the IT security of examined prototypes) and metrological examination of the measurement functionality itself (addressing measurement uncertainty, reproducibility etc.) are two separate tasks usually conducted by two separate examiners. Therefore, it appears justifiable to monitor changes to the protection and security measures, e.g., the order of transitions within internal DFAs, separately from the measurement function itself.

If a critical modification is detected (if-then-statement in the lower right corner of Figure 3), a manual intervention is needed. In order to revert to a certified state, conformity assessment for such a modified instrument must be repeated. If problems are identified with the modified instrument during re-assessment, potential corrective actions regarding improper use of non-conformant measuring instruments may be required. The workflow of the procedure will be illustrated by a detailed example in Section V. Depending on the complexity of the automaton, learning its representation can be computationally expensive. Since measuring instruments usually possess rather simple DFAs and devices like taximeters are usually inactive for longer periods on a daily basis, applying the L^* algorithm to such instruments still appears feasible.

It should be noted that neither L^* nor L^*_M work in actual black-box scenarios. Instead, they require the existence of a teacher T who has full access to the SUT and can answer membership and equivalence queries accordingly, see Section II-A. To this end, it is envisioned that such a teacher T might be developed by the instrument manufacturer and evaluated during initial conformity assessment. Subsequently, the teacher T could then act as a test interface for market surveillance and inspectors, enabling them to continually monitor individual device in the field remotely until the need for manual intervention arises.

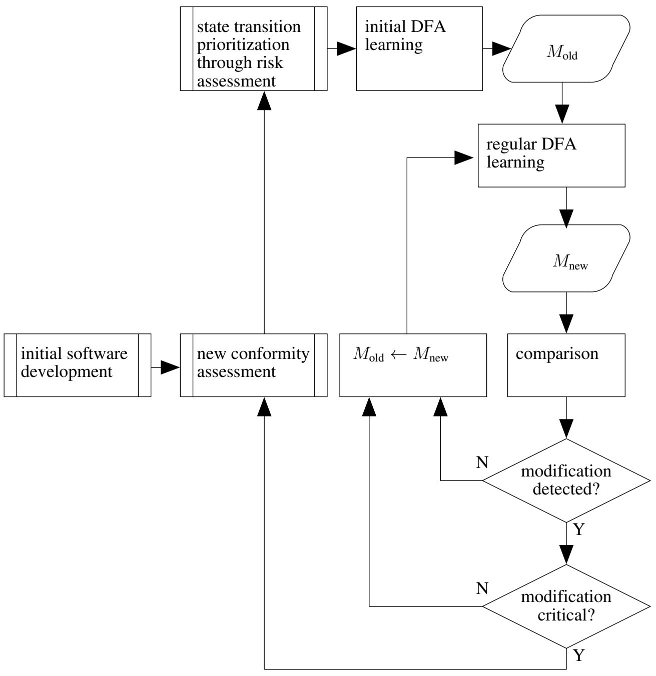
state transition prioritization through risk assessment
initial DFA learning
M_{old}
regular DFA learning
M_{new}
initial software development
new conformity assessment
$M_{\text{old}} \leftarrow M_{\text{new}}$
comparison
N
modification detected?
Y
N
modification critical?
Y

Fig. 3. Anticipated workflow of the risk-based testing approach. The initially learned model M_{old} is continually compared with newly learned models M_{new}, unless the comparison between both models identifies a critical state transition.

V. Exemplary Evaluation

To illustrate the usage of the proposed risk-based ACQC workflow for measuring instruments, a real-world exemplary instrument will be examined in detail in Section V-A, followed by an investigation into different types of new state transitions in Section V-B and a discussion regarding discovery of unknown states in Section V-C. An analysis of the example that also identifies open issues of the approach will be provided in Section V-D.

A. Taximeter as a complex DFA

A taximeter (as defined in Annex IX of the MID [4]) is a ”device [that] measures duration, calculates distance on the basis of a signal delivered by the distance signal generator. Additionally, it calculates and displays the fare to be paid for a trip on the basis of the calculated distance and/or the measured duration of the trip.“ Therefore, the sensor is not part of this type of measuring instrument and it solely performs processing operations on the received digital distance data. This makes

TABLE II
TRANSITION FUNCTION δ FOR THE TAXIMETER EXAMPLE IN FIG 4. STATE F REPRESENTS A FREE VEHICLE WITH NO PASSENGER, O AN OCCUPIED VEHICLE AND M AN ONGOING MEASUREMENT. I REPRESENTS RETRIEVAL OF FISCAL DATA AND U A SOFTWARE UPDATE. SYMBOL s SIGNIFIES THE START OF A MEASUREMENT, e SIGNIFIES EXITING A STATE, i INITIALIZES A FISCAL REVIEW AND u CORRESPONDS TO A SOFTWARE UPDATE PACKAGE.

		symbol			
		s	**e**	**u**	**i**
state	**F**	O	F	U	I
	O	M	F	O	I
	M	M	F	M	M
	I	I	F	U	I
	U	U	F	U	U

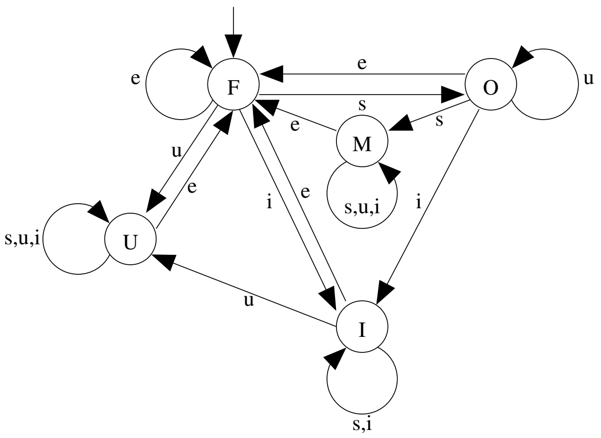

Fig. 4. DFA representing the different states of a taximeter and the state transitions. The taximeter states are F for a free vehicle with no passenger, O for an occupied vehicle, M for an ongoing measurement, I for retrieval of fiscal data and U for a software update. The input alphabet consists of the symbol s to start a measurement, e to exit a state, i to initialize fiscal review and u for a software update package. In the original model of the DFA shown here, the instrument will always return to the default state F after completing a software update in state U.

taximeters especially suitable as a test case for the proposed method, see Section III.

Since taxis have frequently changing customers, they usually possess DFAs that mirror the process of a customer entering and leaving a vehicle as well as the starting and stopping of the measurement itself. Subsequently, said DFAs contain a state F that represents a free vehicle, whereas the DFA enters the state O to signal that the taxi is now occupied. This could either be triggered through a button on the device or by means of a seat contact. For the sake of a simple example, it shall be assumed that the price per travelled kilometer is fixed. It should be noted, however, that some EU member states have complex tariff structures that take current time, number of passengers etc. into account. If the occupied vehicle starts travelling, the internal DFA then enters the measuring state M. To leave the state, the customer must first pay the price, after which the driver pushes the corresponding button to exit the measurement state. In addition, most taximeters also possess an option to retrieve fiscal data, such as the overall total of calculated fares and the complete travelled distance. Both are needed to perform tax audits for taxi companies. The corresponding fiscal inspection state I can be entered if no measurement is running and it should not be possible to start a measurement from this state. Finally, some taximeters possess a functionality to perform software updates. This functionality shall be represented by a state U. A use case oriented input alphatbet would then consist of the symbol s to start a measurement or transition from the free state F to the occupied state O. The symbol e correspondingly signals the exiting of the current state and return to the default free state F. Input symbols i for fiscal inspection and u for a software update indicate the command to either perform an inspection or trigger a remote update. The corresponding graphical representation of the complete DFA may be found in Figure 4. The corresponding transition function δ, which maps a current state to the next state given a specific input symbol, is shown in Table II. As indicated in Section II-D, all measuring instruments must be subjected to a risk assessment as part of the necessary conformity assessment procedure before putting such instruments on the common European market. Figure 1 shows the attack probability tree as one outcome of the risk assessment procedure for a taximeter's software. When comparing the attack probability tree with the example described above, it should become clear that child node B (modification of a taximeter's parameters) cannot be linked to the transition function δ in Table II since there is no corresponding state that enables parameter changes. Child node C (replacing the software of a taximeter), however, could be enabled by inadmissible transitions to and from the update state U. In fact, node E (installing new software) addresses specifically the functionality behind the update state. In this context, one should keep in mind that breaking and replacing of the seal (represented by child nodes F and G) do not necessarily have to address physical hardware seals. So-called electronic seals realized as protected logbooks are equally common in Legal Metrology [5]. Subsequently, all additional transitions to and from the update state U (represented by the detected state su in Table III) would be classified as critical during conformity assessment since such transitions could interfere with normal processing of updates and damage the continuous audit trail of logged software modifications.

TABLE III
TRANSITION FUNCTION δ OBTAINED BY L^* ALGORITHM FOR THE ORIGINAL TAXIMETER EXAMPLE FROM FIGURE 4. STATES ARE GIVEN IN THE REPRESENTATION OBTAINED BY THE ALGORITHM, E.G., ssu, TOGETHER WITH THEIR CLEARTEXT REPRESENTATION, E.G., M.

		symbol			
		s	**e**	**u**	**i**
state	**s/F**	ss	s	su	si
	ss/O	sss	s	ss	si
	sss/M	sss	s	sss	sss
	si/I	si	s	su	si
	su/U	su	s	su	su

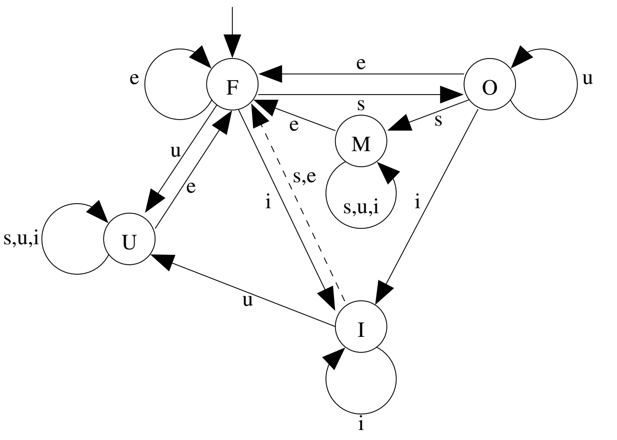

Fig. 5. DFA representing the different states of a taximeter and the state transitions after addition of an non-critical state change from fiscal inspection I to the free state F (dashed arrow). While originally only the input symbol e for exiting triggered that change, symbol s now has the same effect. In the orignal example, symbol s had no effect on the automaton when in state I.

As explained in Section II-B the L^* algorithm produces a transition function δ that references the internal states of the examined DFA by their corresponding input sequences. To improve readability of the example, the first column of Table III contains both the cleartext representation of the states as well as their representations obtained by the learning algorithm, which correspond to the symbol sequences needed to transition to a certain state. Since s is the first symbol tested by thevused algorithm, it denotes the default state F also with that symbol. Consequently, all other state representations start with that symbol, too. Section V-C will address how representation variations may affect the interpretation of the algorithm output and how this effect can be mitigated.

B. Non-critical and critical state changes

To test the proposed automatic detection method, the DFA of the taximeter shall now be modified by adding another transition from state I for fiscal inspection to the free state F triggered by the input symbol s (originally only triggered by symbol e), see Figure 5. Although this transition no longer matches the original assignment linked to that input symbol, it does not constitute a critical modification from the point of view of conformity assessment. Following the learning cycle proposed in Figure 3, the L^* algorithm is applied to the modified DFA resulting in a new version of the transition function δ, see Table IV. As can be seen from the table, the state representation obtained by the L^* algorithm remains the same, e.g., state M is still represented by the input symbol sequence sss. The only difference between the original transition function (see Table III) and the updated version in Table IV may be found in the row for transitions from state si/I, where the input symbol s now triggers a return to state s/F. Since an added transition to this state was deemed uncritical during conformity assessment, monitoring of the system can be continued without the need for human intervention.

TABLE IV
TRANSITION FUNCTION δ OBTAINED BY APPLICATION OF THE L^* ALGORITHM TO THE TAXIMETER EXAMPLE FROM FIGURE 5 WITH A MODIFICATION THAT ENABLES A SECOND TRANSITION FROM I TO F. THE CORRESPONDING NEW STATE TRANSITION IS UNDERLINED.

		symbol			
		s	**e**	**u**	**i**
state	**s/F**	ss	s	su	si
	ss/O	sss	s	ss	si
	sss/M	sss	s	sss	sss
	si/I	$\underline{s}$	s	su	si
	su/U	su	s	su	su

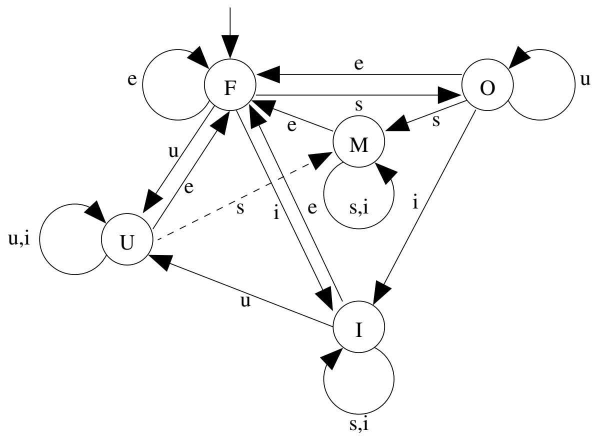

Fig. 6. DFA representing the different states of a taximeter and the state transitions after addition of a critical state change directly from the update state U to the measurement M (dashed arrow) if the input symbol s is received.

As a second test case, the original taximeter DFA shall now be modified by adding a state change between update state U and measurement state M, see Figure 6. Such a transition was deemed critical during initial assessment of the measuring instrument and should trigger an automatic response. The corresponding function δ learned after application of the L^* algorithm to the modified example is given in Table V. Again, the linking between cleartext names of the states and the representations found by the algorithm appears to be unchanged. However, as can be seen from Table V, the transition function δ now also reflects the intended additional transition from the

TABLE V
TRANSITION FUNCTION δ OBTAINED BY APPLICATION OF THE L^* ALGORITHM TO THE TAXIMETER EXAMPLE FROM FIGURE 6 WITH A MODIFICATION THAT ALLOWS SWITCHING TO MEASUREMENT STATE M IMMEDIATELY AFTER A SOFTWARE UPDATE (REPRESENTED BY STATE U).

		symbol			
		s	**e**	**u**	**i**
state	**s/F**	ss	s	su	si
	ss/O	sss	s	ss	si
	sss/M	sss	s	sss	sss
	si/I	si	s	su	si
	su/U	$\underline{sss}$	s	su	su

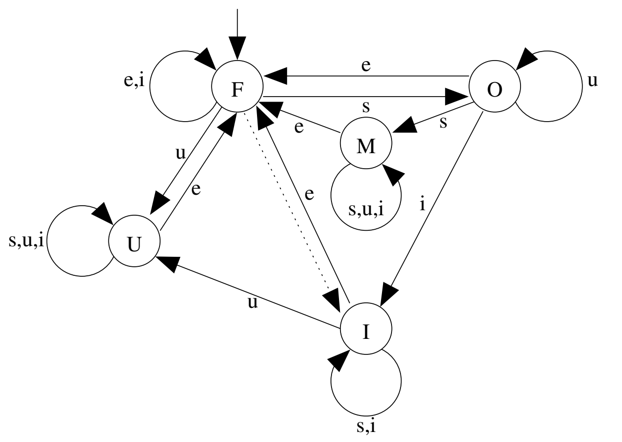

Fig. 7. DFA representing the different states of a taximeter and the state transitions after removing a state transition from free state F to the fiscal inspection state I (dotted arrow). Instead, the DFA remains in state F if an input symbol i is received in that state.

update state U (represented by the symbol sequence su in the table) to the measurement state M (represented by the symbol sequence sss). Since any additional transition to and from the update state was classified as critical during the original risk assessment (see Section V-A), the algorithm now issues a warning that triggers a repetition of the conformity assessment procedure to check whether the modified instrument still complies with legal regulations. As part of the repeated assessment, the risk analysis would also be performed and evaluated again. During this step, the classification of critical state changes might have a different outcome because of additional information not available during original assessment. If the modified software were deemed acceptable, the proposed quality assurance algorithm would be supplied with a new list of critical state changes and the L^* algorithm would be started again. If not, manual withdrawal of all affected taximeters in the field would become necessary.

C. Necessary discovery of state correspondences

As indicated in Sections II-B and V-A, the state representations by their corresponding input symbol sequences within the transition function δ obtained by the L^* algorithm depend on the order in which states are discovered. To illustrate this fact, a modified version of the original taximeter DFA shall be used, where the state transition from free state F to the fiscal inspection state I has been removed, see Figure 7. The corresponding state transitions identified by the L^* algorithm may be found in Table VI. Due to the different order of state discovery, the fiscal inspection state I is now no longer referenced as si but rather as ssi in the table. Since such an assignment of a different label could potentially affect more than one state, it becomes necessary to add a matching step to the comparison step between consecutive learned models $M_{\text{old}}, M_{\text{new}}$ included in the proposed workflow in Figure 3. For the sake of simplicity, the matching step shall consist of checking all possible assignments between cleartext

TABLE VI
TRANSITION FUNCTION δ OBTAINED BY APPLICATION OF THE L^* ALGORITHM TO THE TAXIMETER EXAMPLE FROM FIGURE 7 AFTER DELETING ONE OF THE ORIGINAL STATE TRANSITIONS FROM F TO I.

		symbol			
		s	**e**	**u**	**i**
state	**s/F**	ss	s	su	$\underline{\text{s}}$
	ss/O	sss	s	ss	$\underline{\text{ssi}}$
	sss/M	sss	s	sss	sss
	$\underline{\textbf{ssi/I}}$	$\underline{\text{ssi}}$	s	ss	$\underline{\text{ssi}}$
	su/U	su	s	su	su

representations of DFA states and corresponding symbolic state representations from Table II. The one assignment that minimizes the number of new or modified state transitions compared to the original DFA shall then be assumed to be correct and the identified transitions shall be evaluated against the list of critical state changes from the risk assessment. If there is more than one assignment that minimizes the number of new or modified state transitions, the state assignment is no longer unambiguous and the modification will be assumed to be critical by default. This approach will only fail under two conditions: If the overall number of discovered states does not match the original DFA or if sufficiently many state changes have been implemented by the manufacturer so that the learned transition function matches the original one, even if the underlying functionality is different. Both cases will be revisited in Section V-D.

D. Analysis of the Example

When comparing the transition functions obtained by the L^* algorithm for the non-critical and critical modifications of the taximeter DFA (see Tables IV and V respectively), it can be seen that the proposed risk-based quality control approach can effectively identify and deal with both types of modifications. Manual intervention as the result of a detected assumed critical change will likely be able to assess the actual impact of the modifications and ensure compliance of all serial devices in the field. The monitoring approach might fail, however, if several state transitions are modified or added iteratively so that they are only examined individually by the L^* algorithm. Even if the combination of modifications or additions produces effects that are in violation of legal requirements, the current implementation would not be able to detect these effects. However, this scenario is implicitly already covered by today's practice of performing periodic reverifications of measuring instruments in use. As outlined in Section IV, the manufacturer of the measuring instrument would need to implement a teacher in the form of a test interface for the proposed approach to work. Of course, it cannot be guaranteed that such an interface actually interacts with the internal DFA of the measuring instrument. Instead, a dummy DFA could be implemented to hide software modifications from the automatic quality checker. During the above-mentioned reverifications, however, it would be possible to also practically check whether the implemented teacher T

correctly abstracts the measuring instrument's DFA for the external Learner L^*, thereby mitigating such a threat. As illustrated in the example in Section V-C, reproducibility of the L^* algorithm's output depends on the context-based interpretation of learned state labels. The proposed brute-force matching algorithm to identify correspondences between cleartext state representations and learned state identifiers has several shortcomings which shall be addressed here:

- While a brute-force approach, matching all DFA states against all possible representations, is guaranteed to find one or more optimal matches, the approach might become computationally complex if large DFAs are monitored. Breadth-first search algorithms should be able to provide quicker solutions without missing any transition modifications.
- As discussed in Section V-C, the number of discovered states does not necessarily have to match the number of states in the original DFA, even after application of the DFA minimization algorithm. In such a case, the currently investigated approach would always classify the modification as critical, even if a state has been removed that is not legally regulated.
- It is theoretically possible to implement sufficiently many state changes simultaneously that cannot be detected because the learned transition function δ contains the same number of states and matching state transitions as the original transition function.

It should be noted again that the current approach only focuses on simple state transitions within DFAs while the behavior of more complex instruments than heat meters or taximeters will likely be better characterized by the more general Mealy automata, see Section III. Using Mealy automata would enable checking of input and ouput behavior of such systems, thus ensuring a wider range of useful application scenarios. Investigation into an approach using the adapted L^*_M algorithm will, therefore, form the basis for further work. Similarly, machine learning algorithms such as the one described by Yan, Tang, Luo, Fu, and Zhang in [16] are already able to perform anomaly detection for complex IT systems. Due to the similarities between such systems and measuring instruments, similar approaches might also be able to model and monitor the software of measuring instruments to some extent, while potentially bridging the gap between automata models and mathematical models for measurements themselves. Once more elaborate quality control approaches for software in measuring instruments are available and have proven their reliability, it might be possible to replace mandatory periodic reverifications with risk-based reverifications based on the detected behavior of individual devices. If proven useful, such quality control approaches could be added as an acceptable solution for dealing with software modifications in the currently established technical interpretation of the MID, namely the WELMEC 7.2 Software Guide [5]. Such an acceptable solution could facilitate the uptake of the method and harmonize the approach across the EU if needed.

VI. Summary

In this paper, a new risk-based quality control approach for measuring instruments in legal metrology was proposed as a high-level attempt to realize functional identification for software of such systems. The approach is based on work published in [2] as well as [3] and uses the L^* algorithm to monitor changes in the DFAs of measuring instruments in the field. To this end, the outcome of the mandatory risk assessment procedure for regulated measuring instruments is used to identify critical state transitions to be checked if software changes occur. Based on an example for a DFA in a taximeter, the approach was evaluated regarding the detection of non-critical and critical state changes, even in light of varying conditions like modified state representations. To mitigate potential effects of varying state representations, a brute-force matching algorithm was added to the proposed method that can effectively reduce the number of falsely identified critical state transitions. This proof of concept has shown that automatic quality control of measuring instruments is indeed possible if the SUT fulfills certain preconditions, such as a clear separation between measurement function and internal DFA. Manual intervention in case of doubt and periodic reverifications are still necessary to cover all eventualities. While the method requires instrument manufacturers to implement a test interface in their devices, they would benefit from the possiblity of issuing bugfixes to their software without having to go through conformity assessment by default. Similarly, conformity assessment bodies would have to check said interfaces initially, but would benefit when updates are deemed to be in line with the originally certified instrument functionality, thus avoiding repetition of software examinations. Finally, market surveillance authorities and inspectors in Legal Metrology could use the data provided by the L^* algorithm to assess modifications in devices in the field to a certain extent without the need to be on site. It is envisioned that the approach would work in any industry sector where software systems are used whose compliance with specific requirements must be checked by external authorities in the field. Further work will focus on validating the current approach with additional, more realistic practical test cases (also outside legal metrology) and optimizing the matching algorithm between learned and known state representations. Extending the approach from DFAs to more general Mealy automata will hopefully pave the way towards an actual functional identication mechanism for software in measuring instruments since it would also encompass the output language of devices in the field rather than simply monitor state transitions.

References

[1] M. Jang, *Linux Patch Management: Keeping Linux Systems Up To Date*, 1st ed. Prentice Hall, Jan. 2006. ISBN 978-0132366755

[2] S. Windmüller, J. Neubauer, B. Steffen, F. Howar, and O. Bauer, "Active continuous quality control," in *Proceedings of the International Symposium on Component-Based Software Engineering*. ACM, Jun. 2013. doi: 10.1145/2465449.2465469 pp. 111–120.

[3] J. Neubauer, S. Windmüller, and B. Steffen, “Risk-based testing via active continuous quality control,” *International Journal on Software Tools for Technology Transfer*, vol. 16, pp. 569–591, 2014. doi: 10.1007/s10009-014-0321-6

[4] EC, “Directive 2014/32/EU of the European Parliament and of the Council of 26 February 2014 on the harmonisation of the laws of the Member States relating to the making available on the market of measuring instruments,” European Union, Council of the European Union; European Parliament, Directive, February 2014.

[5] “WELMEC 7.2 Software Guide,” European cooperation in legal metrology, WELMEC Secretariat, Braunschweig, Standard, Mar. 2022.

[6] M. Sipser, *Introduction to the theory of computation*, 2nd ed. Boston, Massachusetts: Thomson, 2006. ISBN 0-534-95097-3

[7] G. H. Mealy, “A method for synthesizing sequential circuits,” *The Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955. doi: 10.1002/j.1538-7305.1955.tb03788.x

[8] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987. doi: 10.1016/0890-5401(87)90052-6

[9] M. Shahbaz and R. Groz, “Inferring mealy machines,” in *FM 2009: Formal Methods*, A. Cavalcanti and D. R. Dams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-05089-3_14. ISBN 978-3-642-05089-3 pp. 207–222.

[10] M. S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis - The CORAS Approach*. 0314 Oslo, Norway: Springer, 2011. ISBN 978-3-642-12323-8

[11] M. Esche, F. Grasso Toro, and F. Thiel, “Representation of attacker motivation in software risk assessment using attack probability trees,” in *Proceedings of the Federated Conference on Computer Science and Information Systems*, Prague, Czech Republic, September 2017. doi: 10.15439/2017F112 pp. 763–771.

[12] ISO/IEC, “ISO/IEC 27005:2011(e) Information technology - Security techniques - Information security risk management,” International Organization for Standardization, Geneva, CH, Standard, June 2011.

[13] ——, “ISO/IEC 18045:2008 Common Methodology for Information Technology Security Evaluation,” International Organization for Standardization, Geneva, CH, Standard, September 2008, Version 3.1 Revision 4.

[14] M. Esche and F. Grasso Toro, “Developing defense strategies from attack probability trees in software risk assessment,” in *Proceedings of the Conference on Computer Science and Information Systems*, 2020. doi: 10.15439/2020F21 pp. 527–536.

[15] “Guide to the expression of uncertainty in measurement - part 6: Developing and using measurement models,” Joint Committee for Guides in Metrology (JCGM), BIPM, Sèvres Cedex FRANCE, techreport, Mar. 2020.

[16] S. Yan, B. Tang, J. Luo, X. Fu, and X. Zhang, “Unsupervised anomaly detection with variational auto-encoder and local outliers factor for kpis,” in *2021 IEEE Intl. Conf. on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking*. IEEE, 2021, pp. 476–483.