Improving Testability of Software Systems that Include a Learning Feature

Lydie du Bousquet
Univ. Grenoble Alpes,
CNRS, Grenoble INP, LIG,
38000 Grenoble, France
email: Lydie.du-Bousquet@imag.fr

Masahide Nakamura
Graduate School of System Informatics,
Kobe University, Japan,
RIKEN AIP.japan
email: masa-n@cs.kobe-u.ac.jp

***Abstract*—In this article, we describe the work we did to validate a case study application that includes a learning feature. Our objective was to express properties that can be used for testing or monitoring the quality of the application, taking to account the learning dimension. To express relevant properties, we had to modify the architecture of the application to add supplementary output. They indicate whether the learning phase is achieved or not, and if the system notices some significant changes in the environmental conditions. Here, we report about the lessons learnt about this process.**

***Index Terms*—Learning System; Validation; Testability; Design for Testing**

I. Introduction

Nowadays, Artificial Intelligence (AI) is becoming more and more important in software systems. Learning ability for new apps or systems is a promise of a personalised experience for every user based on individual preferences. Thus, user's satisfaction is expected to be improved.

For example, authentication facilities are now offered for smart-phones that learn and recognise the user's face [1]. Virtual personal assistants are able to learn the user voice to satisfy the voice commands. AI systems send recommendations for shopping[2]. Self-driving cars are travelling more and more kilometres.

However, safety or security problems are regularly risen for those applications [3]. This makes AI more suspicious for users and increases the need of validation, verification, and even certification [4]. But learning features make software system much more difficult to validate [5]. Indeed, while learning the user's habits or the environmental characteristics, the system behaviour is evolving in an unpredictable way [6]. Specification becomes somehow difficult to express, which impacts the validation procedures.

In this article, we report the lessons learnt from the validation of a case study application that includes a learning feature. To be able to express properties that can be used for testing or monitoring, we had to modify the design of the systems. In Section II, we first report on some related works dedicated to the validation of machine learning algorithms and systems that include such algorithms. We then detail our case study (Section III) and the expression of properties (Section IV). We discuss on the lessons learnt in Section V.

II. Machine learning and validation

A. Validation of the algorithms

There is a large variety of Machine Learning (ML) algorithms dedicated to different applications. For example, classification ones are used to classify discrete inputs to predefined categories. Clustering algorithms are used to group similar inputs (into clusters). Regression analysis is used for prediction and forecasting.

A ML algorithm is expected to predict well after training. This property is called generalisation. The first step of the validation is thus to check the quality of the prediction, also called *performance* of the ML algorithm. This can be compared to the functional validation step in software engineering, which aims to ensure that a program is doing well what it is supposed to do.

Performance measures are specific to the algorithm classes that is considered. For pattern recognition and binary classification, this is evaluated through *precision*, *recall* and *F-measure* (the weighted harmonic mean of precision and recall) [7]. They are based on the count of true/false positive/negative verdicts. True (resp. False) verdict are used to differentiate when the answer of the system recognises (or not) the input. Positive (resp. negative) verdict states whether the answer is correct or not. For algorithms such as classification and regression, generalisation error can be evaluated to measure how accurately the algorithms are able to predict outcome values for previously unseen data [8].

Other properties can also be expected for ML algorithms. *Stability* evaluates how much a ML algorithm is perturbed by small changes to its inputs: the predictions of a stable algorithm will not be very affected if the inputs change just a little bit [9]. For a large class of learning algorithms, notably empirical risk minimisation algorithms, certain types of stability ensure good generalisation. *Training speed*, *efficiency*, *compactness* are also dimensions that can be evaluated, especially to compare different algorithms.

To evaluate the quality of the algorithms, several methods are proposed. In holdout evaluation, data available for training are split into two sets, one for the training itself, the second

one for the validation [10]. This can be done randomly or may involve more complex sampling methods.

Cross-validation denotes a set of more sophisticated validation methods. Basically, they consist in dividing the dataset into equally sized groups of instances (called folds). The learning algorithm is then applied several times, each time using the union of all subsets but one, which is used as a test set [11]. The cross-validation methods are different from one another by the way to build the folds and to use them. To produce the learning set, some test methods can be applied [12], [13].

Learning and validation with cross-validation approaches can give very good results. But they are mainly dedicated to situations *"in the lab"*. For *"in the wild"* cases, i.e., for real world situations where applications learn from the final user, simpler approaches might be applied due to lack of time or data. Poorer results are observed [14]. For this reason, it is necessary to consider also the quality of the final system considered as a whole.

B. Validation of the final system

Validation of a system that includes some learning feature is difficult. As said previously, the final usage of the application is not easily predictable since it varies with respect to the environment. Sometime, even for one specific user, her needs may evolve with the time. This makes *specification* impossible to express with precision [5] [15] . For the same reasons, it is usually not possible to assess precisely the *environment* characteristics of the final system.

In [16], authors use *simulations* to evaluate the quality of an online adaptive system for electric wheelchairs that learn to avoid obstacles. For this specific application, some safety properties are easy to express, e.g., the wheel chair should not enter in collision with any obstacles. But quality of the learning is more difficult to assess with a property. So, authors evaluated manually the path smoothness of the wheel chair in different environments, based on a *comparison* of two algorithms.

In [17], authors focus on the validation of learning features embedded in applications dedicated to intelligent inhabited environments. They performed experiments in which an intelligent application learned and adapted itself to the user behavior, while she stayed in a real equipped flat for five days. Here again, the quality of the learning feature is assessed by a comparison of the results of different algorithms.

Model-checking approach has been used in [18]. Authors focus on the safety and robustness validation of vision systems that can be used for self-driving cars. One of the considered safety properties is that "self-driving cars steering angle should not change significantly for the same road under different lighting conditions". To evaluate this invariant, authors propose a framework that transforms a given image with different transformation functions (e.g., rotation). The objective is to assess the quality of the final application (after learning) by analysing how often the invariant is violated.

In [19], authors advocate the usage of quantitative verification at run-time to identify and even predict requirement violations, in addition to off-line verification for self-adaptive feature, but it is not clear how to adapt them for specifying properties on learning.

In [20], authors advocate the usage of metamorphic testing to ease the problem of the oracle expression ML applications. Metamorphic testing aims at creating new test cases from the existing ones thanks to a transformation. Well chosen, the transformation approximates the expected outputs of the new tests based on the expected outputs of the old ones.

Beyond the expression of the expected properties (and/or the metamorphic relation), system testing remains difficult to apply, especially because it requires to control the input of the system [21]. For instance, testing an intelligent home application that regulates the home temperature may require to be able to modify the home temperature by other means, in order to check that the system under test reacts correctly. Overheating a room in winter in order to check that the air-conditioner can cool it correctly is often unacceptable. .

For this reason, the final system is often tested in a simulated environment before deployment [22]. For a validation after deployment, monitoring is often preferred [23] [24]. It consists in observing the outputs of the system without controlling the inputs.

C. A software engineering viewpoint of testability

Software testing is the process of executing a program with the intent of finding errors [25]. It has emerged as one of the major techniques to evaluate the implementation reliability. Unfortunately, testing is usually an expensive process. It can represent more than 40% of the total cost of the software development [26].

Testability denotes the ability of a system to be tested [27]. Originally, testability was defined for hardware components. For software systems, several definitions have been proposed. In [28], testability is defined as the effort needed for testing. For Binder, testability is the relative ease and expense of revealing software faults [29]. Other definitions allow a quantitative evaluation of the testing effort [30] or represents the probability to observe an error at the next execution if there is a fault in the program [31].

Being able to characterise and to produce testable systems has become a preoccupation more and more important for software companies. Basically, it often means to increase the ability to observe the internal behaviours of the system under test (observability) and/or to increase the ability to control the system under test (controllability).

The case study that follows was a source of reflection for the expression of properties that can be used for testing oracle or monitoring. To express them, we had to modify the application design. In the following, we first describe the case study. We then show how the expression properties impacted the design.

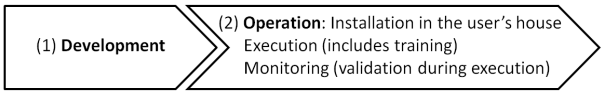

FIG. 1. ABSTRACT SOFTWARE LIFE-CYCLE OF THE INTELLIGENT AIR-CONDITIONNER

III. CASE STUDY

A. The application under test

Our case study is an "intelligent air-conditioner" (iAC). It controls a classical air-conditioner and offers a planning functionality. Thanks to this last one, the user can choose a timer value above which the room temperature is expected to be equals to a chosen one. To make this possible, the system includes an intelligent feature that learns how long it takes to warm or to cool the room (room thermal inertia).

The life-cycle of the iAC software is depicted at an abstract level in Figure 1. It includes two parts, which denote the time before and after the deploiement of the application. The learning phase is carried out once the iAC is installed in the user house.

We chose this case study because it is an example of a system "in the wild": it is not possible to achieve training on an pre-existing data set nor is it possible to apply cross validation methods. It is also difficult to achieve testing at system level (i.e., on a real installation) because the environment of the system is hardly controllable for the reasons given Section II-B.

Our objective is thus to achieve monitoring of the system execution in order to check that the system behaviours are adequate. By monitoring, we mean that a program will periodically check that the inputs and outputs of the iAC satisfy the expected behaviours. We especially want to detect if the learning phase carried out after the installation leads to inacceptable behaviours (to prevent situations such as Tay chatbox [6]). The difficulty relies in expressing what are the adequate behaviours, i.e., to express the right specification/properties. In the following, we report on the work done to elaborate those properties, which takes place within the development phase of the lifecyle.

Intuitively, the user expects that the the planing functionality works correctly (the required temperature should be reached on schedule). This means that the learning feature should learn correctly the thermal inertia. Of course, at the beginning, the system can fails, but the error should be less and less with the time. The user could expect that the learning does not last too long. Another important requirement of the planning functionality is that it is supposed to spend as less energy as possible to achieve the chosen temperature: i.e., it should switch-on the AC just on time w.r.t. the room thermal inertia.

The classical air-conditioner can be switched on and off. It has three modes: `Cooling`, `Heating` and `Idle`. The AC enters the `Cooling` mode if the observed temperature is beyond the required temperature of more than one degree. It enters in the `Heating` mode if the observed temperature is below the required temperature of more than one degree. It enters in the `Idle` mode when the observed temperature is equal to the required one $\pm$ 0.5. The required temperature can only be set between 17 and 27^{o}C.

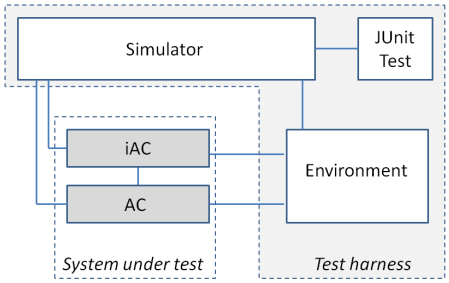

FIG. 2. SIMPLIFIED TEST HARNESS

For the case study, both classical and intelligent air-conditioners were developed in Java. The learning feature was implemented as an ad-hoc algorithm. It has no importance by itself, since the system will be considered as a black-box for the validation point of view [21].

B. The validation harness

As said previously, our case study aims at providing a support to express some expected properties of a system with learning abilities. For this reason, the validation harness is very simple. It consists of a simulator and an environment component, which are controlled by a JUnit test (see Figure 2).

The environment component aims at simulating the room temperature evolution, which is read by the temperature sensors of AC and iAC. It is possible to modify the room inertia rules during the simulation to fake some environment changes (e.g., outside temperature, opened windows).

The simulator is responsible for the validation progress. It includes:

- *initialisation* methods, to create, initialise and connect the iAC, AC and environment components,
- *property* methods (also called oracle methods), in which expected properties of the system under test are expressed
- a "step" method that deals with the evolution of the system for a given time lapse. During one "step", the simulator checks the state of the AC and asks the environment to update accordingly equation (1).

$$temp = \begin{cases} temp + \delta_1 \texttt{ if AC is On and Heating} \\ temp - \delta_2 \texttt{ if AC is On and Cooling} \\ temp - \delta_3 \texttt{ if AC is Idle or Off} \end{cases} \quad (1)$$

The simulator is solicited with a JUnit test file, in which it is possible to specify a modification of δ_1, δ_2 and δ_3 during the simulation. In this test file, it is also possible to evaluate the value of the oracle properties as JUnit assertions. Thus, if one of these properties is violated during the simulation, a `Fail` verdict is raised by JUnit.

IV. Expressing properties

Our intelligent AC has to achieve the basic properties expected for an AC. Informally, this means that when the AC is `On`, "it should heat (resp. cool) the room when it is in its `Heating` mode (resp. `Cooling` mode)", and "it is supposed to be in the `Heating` (resp. `Cooling`) mode when the room temperature is substantially below (resp. above) the expected one".

In addition, the iAC has to manage correctly the timer feature: the expected temperature should be reached on time. Moreover, it is supposed to spend as less energy as possible to achieve the chosen temperature. In the following, we focus on the validation of the intelligent feature.

Let us first consider P, a property stating that the room temperature should be equal to the expected one when the timer is elapsed. Considering the imprecision of the measure, such a property can be expressed as:

$$(P): \texttt{iAC.timerElapsed} \Rightarrow \\ |\ \texttt{env.temp} - \texttt{iAC.requiredTemp}\ | < 0.5$$

where `iAC.timerElapsed` is a Boolean variable that is `true` exactly when the time is elapsed and `false` otherwise, `env.temp` is the observed environment temperature, and `iAC.requiredTemp` is the user required temperature. The imprecision tolerance was arbitrary fixed to 0.5 here without a loss of generality.

To check if the system spends as less energy as possible, it is possible to monitor how often the AC switches from the `Idle` state to an *active* one (`Heating` or `Cooling`), from the moment where the timer is set until it is elapsed. To compute this, we added the `sim.iSwitch` attribute in the simulator component. It is computed like an observer property. Ideally, `sim.iSwitch` value should be 0 when the time is elapsed, but some flexibility could also be acceptable. Indeed having too restrictive properties could provoke unnecessary fail verdicts. For this reason, we chose the following property.

$$(Q): \texttt{iAC.timerElapsed} \Rightarrow \texttt{sim.iSwitch} \leq 1$$

Moreover, it would be inefficient from an energetic point of view to heat the room just after heating it (or conversely). To capture those situations, in an identical way than previously, it is possible to monitor the number of switches from `Heating` to `Cooling` states (and conversely), and to express an property on the maximum number of changes while a timer is active.

One problem with the previous properties is that they can raise false negative verdicts, i.e., they can be violated even if the system is correct. Three situations have been identified during our tests: (1) the user tries to fix an unfeasible timer, (2) the environmental characteristics have changed during the execution (e.g., a door or window which had been left open), or (3) the training of the system is not achieved.

To fix situation (1), we modify the iAC program, in order to make timer activation possible only when the system evaluates that it has enough time to reach the required temperature within the delay. If it has not, a notification is sent to the user and the AC is switched-on immediately. This has no impact on the previous properties, because the timer is not activated in this situation.

To fix situations (2) and (3), we need to know whether the training is achieved and if the environmental conditions have changed during the execution. In order to get that information, we modify the system design in order to have two new Boolean outputs. The first one, `iAC.stillLearning`, is `true` as long as the system considers that its training is not achieved. The second one, `iAC.envModification`, is `true` if the system noticed significant changes in the environmental conditions while it was trying to achieve the timer requirement, and `false` otherwise.

Thanks to these two outputs, it is possible to rewrite P and Q properties so that a failure occurs only if they are violated after the training end and if the environmental characteristics are the same as the one which were learnt.

$$(P'): \texttt{not}\ P \Rightarrow (\ \texttt{iAC.stillLearning} \vee \texttt{iAC.envModification}\ ) \\ (Q'): \texttt{not}\ Q \Rightarrow (\ \texttt{iAC.stillLearning} \vee \texttt{iAC.envModification}\ )$$

It is worth noting that `iAC.stillLearning` is an important feedback about the system's learning ability. It allows expressing several properties about the quality of the learning feature. For instance, if the system stays in the state `iAC.stillLearning` after using the timer feature "a lot of" times, it may denote a difficulty. Of course, the acceptable learning time (i.e., number of activations) has to be defined. Let `iAC.nbActivation` be the number of activations of the timer since the iAC installation. The following L property is a way to express a too long training process (5 being chosen arbitrarily):

$$(L): (\ \texttt{iAC.nbActivation} > 5\ ) \wedge \texttt{iAC.timerElapsed} \\ \Rightarrow \texttt{not iAC.stillLearning}$$

Similarly, `iAC.envModification` can be used to evaluate the learning feature quality. If it is `true` "too often", it may denote that the system is not able to predict correctly the environment behaviours. In the case of our case-study, the room thermal inertia is susceptible to strongly depends of the door's state (i.e., if it is open or close). If the system is not aware of the door status, it may be not able to learn properly the thermal inertia. The analysis of the `iAC.envModification` variations is a possible way to detect such a situation.

It can be noticed that a system that refuses any timer will be correct with respect to the previous properties (because `iAC.timerActivated` will never be set to `true`). To detect such behaviours, we modified the simulator. Each time a timer is set, the simulator captures the notification of the system (acceptance or rejection of the timer). If the system accepts the timer, nothing is done: the previous properties will

catch abnormal situations. If the system refuses the timer, the simulator starts a specific timer counter.

If the expected temperature is reached at the end of the timer in normal conditions (not (iAC.stillLearning $\vee$ iAC.envModification)), the system may have been too pessimistic, and a counter value is incremented. It is then possible to express a property that if false if the pessimistic refusal rate is greater than a given threshold.

V. Lessons learnt

With this work, our objective was to assess the quality of the learning feature when embedded in the final system. In this context, the issue is to guarantee that the intelligent part of the system does exactly what it is expected to do: i.e., to provide the right indications to take decisions and/or that it carries out the appropriate actions.

Being able to validate the learning feature is tricky because it is not possible to anticipate the usage and the environmental conditions of the system under consideration. Moreover, classical validation methods such as cross-validation or output comparison of several algorithms may not be possible to achieve. Monitoring the system behaviour is possible as long as expected properties can be expressed.

In order to evaluate what kind of properties it is possible to express to qualify learning, we carried out a case study. Beyond the specificity of the considered application, it has been possible to elaborate general lessons.

The system is supposed to learn from its environment and/or from its users, but sometimes, conditions are changing, either occasionally or for a long time. It is important to design the system in such a way that it can (1) detect those changes and (2) provide feedback when they occur.

This has two advantages. First, it is possible to use this feedback to express more accurate properties, and thus limit the number of false negative verdicts in a process of testing or monitoring. Second, it is possible to use this feedback to detect inappropriate behaviour of the learning feature (for instance when training lasts too long or when the system always concludes that the conditions are changing).

Thus, designing a system with learning feature should not consist only in inserting an algorithm upon an existing system. One has to think about what kind of properties the new feature has to satisfy, after what the design should include the outputs necessary to evaluate them. This is necessary not only for the validation step, but also to achieve transparency and accountability of the system [32]. This approach is called "design for testability" in software engineering [33].

To judge the quality of a learning feature, it seems to us that safety and liveness property were a little bit too restrictive. We needed to be able to consider the evolution of different attributes during a period of time, from a statistical point of view, in order to capture suspicious behaviours. Typically, one could observe by this means the user satisfaction. As underlined in [15], the user might be the only possible oracle to judge the quality of the system outputs. If the user satisfaction is also collected, it will be possible to detect when it is mostly negative, and thus denote some learning troubles or inadequacy of the system w.r.t. the needs.

VI. Conclusion and perspectives

In this article, we report on a case study carried out to evaluate how a learning feature can be validated when embedded in a more global system. We chose to apply monitoring approach, consisting in observing the outputs of the system during the execution to detect inconsistent behaviours. The difficulty was to express properties able to detect relevant failures related to the learning process.

The most important conclusion we have from this work is that the expected properties of the system should be considered during the design. While doing that, outputs necessary to evaluate those properties have to be included, otherwise, the final validation would be difficult.

As perspective, we are considering other use cases, in order to see if some pattern of properties to assess quality of learning can be detected.

Notes and Comments: This work has been funded by the project CNRS-PICS 6999 and LIG emergence iCASATE project.

References

[1] M. deAgonia, "Apple's Face ID [The iPhone X's facial recognition tech] explained," *Computerworld*, 11 2017, https://www.computerworld.com/article/3235140/apple-ios/apples-face-id-the-iphone-xs-facial-recognition-tech-explained.html[Retrieved August, 2018].

[2] NewsDesk, "NTT develops AI system to match books with kids," *Asian News Network*, 23rd April 2018, http://annx.asianews.network/content/ntt-develops-ai-system-match-books-kids-71412 [Retrieved August, 2018].

[3] C. S. Smith, "Alexa and Siri Can Hear This Hidden Command. You Cant," *New York Times*, 10th May 2018, https://www.nytimes.com/2018/05/10/technology/alexa-siri-hidden-command-audio-attacks.html [Retrieved August, 2018].

[4] D. Wakabayashi, "Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam," *New York Times*, 19th March 2018, https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html [Retrieved August, 2018].

[5] SOGETI, "Testing of Artificial Intelligence - AI Quality Engineering skills - An introduction," Dec. 2017, https://www.sogeti.com/globalassets/global/downloads/reports/testing-of-artificial-intelligence_sogeti-report_11_12_2017-.pdf [Retrieved August, 2018].

[6] P. Lee, "Learning from Tays introduction," *Official Microsoft Blog*, 25th May 2017, https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/ [Retrieved August, 2018].

[7] T. J. Lee, J. Gottschlich, N. Tatbul, E. Metcalf, and S. Zdonik, "Precision and recall for range-based anomaly detection," *CoRR*, vol. abs/1801.03175, 2018.

[8] D. Mahajan, V. Gupta, S. S. Keerthi, S. Sellamanickam, S. Narayanamurthy, and R. Kidambi, "Efficient estimation of generalization error and bias-variance components of ensembles," *CoRR*, vol. abs/1711.05482, 2017.

[9] S. Mukherjee, P. Niyogi, T. A. Poggio, and R. M. Rifkin, "Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization," *Adv. Comput. Math.*, vol. 25, no. 1-3, pp. 161–193, 2006.

[10] C. Sammut and G. I. Webb, Eds., *Holdout Evaluation*. Boston, MA: Springer US, 2017, pp. 624–624.

[11] ——, *Cross-Validation*. Boston, MA: Springer US, 2017, pp. 306–306.

[12] A. Ramanathan, L. L. Pullum, F. Hussain, D. Chakrabarty, and S. K. Jha, "Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision," in *DATE*. IEEE, 2016, pp. 786–791.

[13] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," *CoRR*, vol. abs/1708.08559, 2017.

[14] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. L. Traon, "Empirical assessment of machine learning-based malware detectors for android - measuring the gap between in-the-lab and in-the-wild validation scenarios," *Empirical Software Engineering*, vol. 21, no. 1, pp. 183–211, 2016.

[15] A. Groce, T. Kulesza, C. Zhang, S. Shamasunder, M. M. Burnett, W. Wong, S. Stumpf, S. Das, A. Shinsel, F. Bice, and K. McIntosh, "You are the only possible oracle: Effective test selection for end users of interactive machine learning systems," *IEEE Trans. Software Eng.*, vol. 40, no. 3, pp. 307–323, 2014.

[16] R. Kurozumi, K. Tsuji, S. Ito, K. Sato, S. Fujisawa, and T. Yamamoto, "Experimental validation of an online adaptive and learning obstacle avoiding support system for the electric wheelchairs," in *SMC*. IEEE, 2010, pp. 92–99.

[17] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments," *IEEE Trans. Systems, Man, and Cybernetics, Part A*, vol. 35, no. 1, pp. 55–65, 2005.

[18] K. Pei, Y. Cao, J. Yang, and S. Jana, "Towards practical verification of machine learning: The case of computer vision systems," *CoRR*, vol. abs/1712.01785, 2017.

[19] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[20] C. Murphy, G. E. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *SEKE*. Knowledge Systems Institute Graduate School, 2008, pp. 867–872.

[21] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2008.

[22] L. du Bousquet, M. Nakamura, B. Yan, and H. Igaki, "Using formal methods to increase confidence in a home network system implementation: a case study," *ISSE*, vol. 5, no. 3, pp. 181–196, 2009.

[23] G. Boracchi, M. P. Michaelides, and M. Roveri, "A cognitive monitoring system for detecting and isolating contaminants and faults in intelligent buildings," *IEEE Trans. Systems, Man, and Cybernetics: Systems*, vol. 48, no. 3, pp. 433–447, 2018.

[24] S. Yerramalla, B. Cukic, M. Mladenovski, and E. Fuller, "Stability monitoring and analysis of learning in an adaptive system," in *DSN*. IEEE Computer Society, 2005, pp. 70–79.

[25] G. Myers, *The Art Of Software Testing*. Wiley-Interscience, 1979.

[26] A. M. Salem, K. Rekab, and J. A. Whittaker, "Prediction of software failures through logistic regression," *Information & Software Technology*, vol. 46, no. 12, pp. 781–789, 2004.

[27] V. Garousi, M. Felderer, and F. N. Kilicaslan, "What we know about software testability: a survey," *CoRR*, vol. abs/1801.02201, 2018.

[28] R. Bache and M. Mullerburg, "Measures of testability as a basis for quality assurance," *Software Engineering Journal*, vol. 5, no. 2, pp. 86–92, 1990.

[29] R. V. Binder, "Design for testability in object-oriented systems," *Communications of the ACM*, vol. 37, no. 9, pp. 87–101, Sep. 1994.

[30] Institute of Electrical and Electronics Engineers, "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries," IEEE, New York, USA, Tech. Rep., 1990.

[31] A. Bertolino and L. Strigini, "On the use of testability measures for dependability assessment." *IEEE Trans. Software Eng.*, vol. 22, no. 2, pp. 97–108, 1996.

[32] V. Dignum, "Accountability, responsibility, transparency - the ART of AI," in *ICAART (1)*. SciTePress, 2018, p. 7.

[33] R. V. Binder, "Design for testability in object-oriented systems," *Commun. ACM*, vol. 37, no. 9, pp. 87–101, 1994.