A Fully Automated and Configurable Cost-Aware Framework for Adaptive Functional Diagnosis

Cristiana Bolchini and Luca Cassano
Politecnico di Milano

FUNCTIONAL TEST/DIAGNOSIS is used when structural approaches are not suitable due to the system complexity [1] and for assessing the correctness of the overall system after testing all the components (with structural techniques). Specifically, functional diagnosis identifies the cause of a failure of an electronic system by applying a set of input stimuli (tests), observing the system responses and comparing them with the expected ones, without any knowledge of the system internal structure. Figure 1 sketches a typical scenario where functional diagnosis is used to identify the cause of a problem, either detected during the postmanufacturing testing, or occurred during the lifetime of the system.

Because of the complexity of modern electronic boards, often the effectiveness of diagnosis is more affected by the experience and the skills of the test/diagnosis team than by the used CAD tools; therefore, a strong con-tribution from the user is usually necessary to drive the diagnosis process. To tackle this problem, a number of "intelligent" techniques have been proposed in the last years.

Research proposals related to board-level functional diagnosis fall into two main classes: design-time and runtime techniques. The former, such as the ones presented in [2], [4], and [5], aim at extracting a model of the board under analysis starting from previously executed diagnoses. They focus on extracting the most accurate model from the smallest amount of historical data. Such a model is then used at runtime to determine the faulty component in a "traditional" way. When a faulty system is found, all available tests are executed and their outcome (i.e., the syndrome) is interpreted with respect to the available model (either extracted by means of the previously mentioned approaches or provided by the engineers) to identify the faulty component. The work in [6] tries to further optimize the model by identifying redundant tests.

Runtime solutions focus on reducing the number of executed tests required for the final diagnosis. In [7], Amati et al. introduce the concept of adaptive incremental diagnosis, i.e., the approach that incrementally executes (groups of) tests and, based on

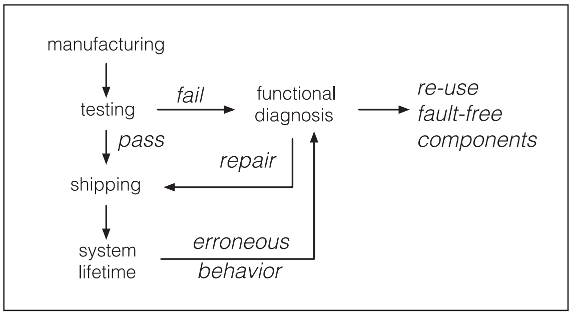

Figure 1. Scenario where functional diagnosis is exploited.

their actual outcomes (referred to as a partial syndrome), adapts the execution order of the remaining tests, and interrupts the diagnosis process as soon as the faulty component can be identified.

This quest for enabling adaptability in test/diagnosis to reduce costs and increase yield has also been highlighted by the International Technology Roadmap for Semiconductors. Moreover, since there can be constraints on the effort to be devoted to the diagnosis (related to tests/components' costs) we deem it interesting to enable the user to configure the diagnosis process to be interrupted when a reasonable result has been achieved in terms of a restricted subset of potentially faulty components.

To pursue such goal, we propose an automated and configurable cost-aware framework for adaptive incremental board-level functional diagnosis, exploiting machine learning (ML) techniques to drive the diagnosis process. The innovative contribution is the integration of the tests and components' cost information and its adaptable exploitation, so that the user may accept a lower accuracy in considering possibly healthy components as faulty, provided their overall cost is acceptable, and the savings in terms of nonexecuted tests is relevant. The framework has a twofold role: on the one hand to automatically drive the diagnosis process by adaptively executing tests based on the incremental collected outcomes and by determining whether the process can be interrupted because the faulty component(s) is identified (dynamic analysis) also leveraging accuracy for costs; on the other hand, to provide feedback on the discriminating power of the available test set and on the isolation capability of the components (static analysis), thus allowing designers to modify the test suite to increase the diagnostic resolution.

Functional diagnosis of complex electronic boards

After the assembling of all the components, an electronic board has to go through a number of testing/diagnosis activities to determine if the system is fault-free. The board is first analyzed by automated X-ray/optical inspection, and then by in-circuit structural test. These two tests rely on the knowledge of the internal structure of the components and allow one to detect a variety of defects, such as open/short circuits, or thinning of the solder. Finally, functional testing is performed to detect faults not targeted by structural tests, using patterns that verify whether the input/output behavior is as expected.

If functional test highlights a problem, functional diagnosis is performed to identify the cause, that is, the faulty component. This activity is driven by multiple goals: 1) identify the faulty component with the highest accuracy in case of expensive boards for which a rework is economically viable; 2) identify fault-free expensive components to be extracted and reused, in boards where a rework is not economically sustainable but components are expensive and it would be anti-economical to throw them away; and 3) keep track of the components' failure rates from various vendors.

Several techniques have been proposed [2], [3], [4], [9] for generating an accurate model of the board under analysis, starting from a limited and possibly incomplete log of syndromes and associated faulty components, using decision trees (DTs) in [2] and the Dempster–Shafer theory in [3]. In [4], three different techniques are presented, exploiting artificial neural networks, support vector machines (SVMs) and weighted majority voting between these two techniques. An approach based on SVMs for an incremental learning of the component–test relation is presented in [9].

Once the accurate model is built, "traditional" runtime diagnosis processes execute all the available tests and analyze the complete syndrome to identify the faulty component.

Adaptive incremental diagnosis has been introduced in [7]; it executes the available tests incrementally, adapting the behavior of the diagnosis engine to the outcomes it collects. At each step, only a subset of the available tests is used, and based on their outcome and of all the previously executed ones, the process may be interrupted (by means of a so-called stop condition) identifying the faulty component, if enough information has been collected. The technique proposed in [7] relies on the Bayesian belief networks to build a model representing the probabilistic relation between failing tests and faulty components. Given a partial syndrome the technique determines the probability of each component to be the faulty one and based on such information, the engine empirically decides to interrupt the process or selects the next test to be executed. The process required the diagnosis engineer's experience. To overcome such limitations, an engine based on data mining has been proposed in [10], and one on statistical data in [12] while a comparative analysis of different ML-based engines is presented in [11].

In the context of an incremental diagnosis approach, where not all available tests are executed, to save time/effort, it is worth noting that, while the goal of all the previously proposed adaptive incremental approaches is achieving a complete accuracy, the framework here presented aims at enabling the user to tune the diagnosis process to meet cost constraints while keeping the desired level of accuracy. The difference between the approaches presented in [2]–[4] and [9] and the present work is that here we focus on driving the runtime diagnosis process and minimize the number of executed tests, while the other approaches focus on the generation of an accurate and efficient system model at design time.

Proposed diagnosis approach

The structure of the proposed framework is shown in Figure 2. It takes as input a model of the board, which considers the components and the tests designed for them and it expresses the probabilistic relationships between faulty components and failing tests. The framework is composed of a static analyzer and a runtime analyzer.

The static analyzer is used at design time, as soon as a model of the board is defined. The goal is to provide feedback to designers about the diagnostic resolution that the set of tests can provide, about the components that could not be isolated and about the tests that would rarely be applied. With this information the designers can improve the model to be exploited at runtime.

The runtime analyzer is applied when the actual diagnosis process has to be performed on a real board after a misbehavior has been observed. It drives the actual diagnosis process by identifying a test execution order and by determining when to stop the process because the faulty component(s) can be identified while also determining components that are surely fault-free. Details about the two engines upon which the runtime analyzer relies can be found in [10] and [12].

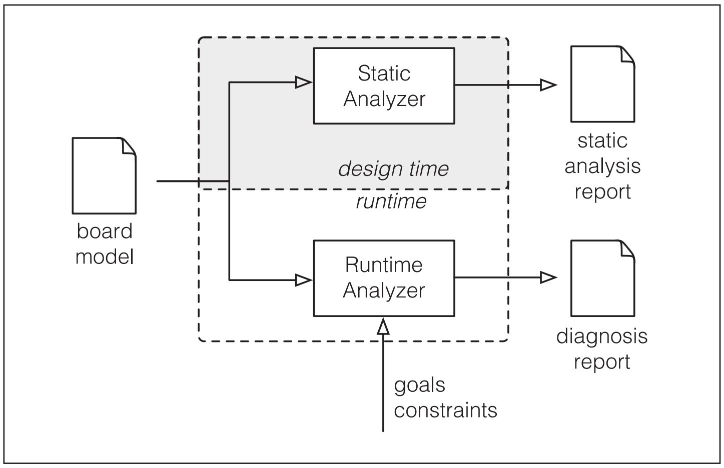

Figure 2. The proposed framework.

Once the user configures goals and constraints, the framework acts autonomously, not requiring any further knowledge/information.

The system model

Electronic boards are generally composed of several components and for each component a number of tests have been typically defined at design time, to be applied directly to the inputs of the component and are designed to fail when the associated component is faulty, and to pass otherwise. Nevertheless, given the complexity of modern electronic boards, when tests are applied to the inputs of the board, it is possible that a test designed for a given component does not even fail when the component is faulty or that a test designed for a given component fails when a different component is faulty because of controllability/observability issues in the board. In this context, test engineers can only qualitatively estimate the probability that given a component being faulty, a test will fail. Based on these considerations, we adopt the model presented in [7] (similar to that used in [8]) where electronic boards are represented as a components–tests matrix (CTM), where each entry $\mathrm{ctm}_{i,j}$ represents the probability that test T_j fails when component C_i is faulty (taking into account the fact that masking effects may occur, for instance, because of interactions with other components).

The static analyzer

The structure of the static analyzer is depicted in Figure 3: it consists of the syndromes analyzer and the tests analyzer. The former determines what components are never isolated, i.e., components that can never be individually identified as faulty, and what components are difficult to diagnose, e.g., identified by a combination of fail/pass that have a low probability of occurrence, based on the given CTM. Furthermore, the tests analyzer determines what tests would rarely be executed at runtime in the incremental approach, because they provide relatively low discriminating power. By exploiting this information, the designers could add tests to increase the diagnostic power of the test set itself or remove tests without compromising the diagnosis accuracy.

Syndromes analyzer

The syndromes analyzer generates the list of all the legal syndromes for the CTM under analysis (not legal syndromes are those that can occur only if more than one component is faulty or if no test fails). For each legal syndrome, the associated faulty candidate component(s) are identified. At this point, it is possible to identify the not isolated components (NICs) (i.e., those components that never appear as the only faulty component associated with a syndrome).

Another piece of information that is gathered from the system model is the occurrence probability of each one of the legal syndromes. Given the combination of tests that may pass/fail when a component is faulty, each combination (i.e., syndrome) has a given probability to occur computed on the $ctm_{i,j}$ values specified in the CTM. Such information is used to lead the automatic engine toward the most probable situations, without compromising the possibility of less frequent cases.

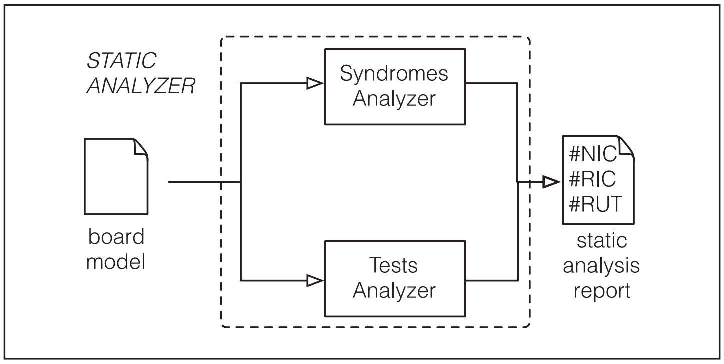

Figure 3. Architecture of the static analyzer.

The static analyzer identifies the rare syndromes, i.e., those syndromes having occurrence probability lower than a configurable value p_{rare}. The components appearing as faulty candidates in rare syndromes only are identified as rarely isolated components (RICs).

Tests analyzer

It relies on a DT representing the information putting into relation failing/passing tests with the faulty components as a binary tree. In more detail, each internal node of the tree represents a test, while leaves represent faulty candidate components; branches between nodes represent test outcomes, i.e., PASS or FAIL: thus each path from the root node to a leaf represents a syndrome with the associated faulty candidate. The DT can be built exploiting either a log of previously performed testing activities, containing <syndrome, faulty candidate(s)> pairs, or the CTM model itself. Once the tree has been built, each path of the tree is traversed starting from the root, while the ratio between the number of visited nodes and the total number of available tests is lower than a configurable percentage $n_{\max_t}$. All visited tests will be considered as essential tests, i.e., tests that cannot be discarded from the CTM, while all the other tests will be considered as rarely used tests (RUTs).

Runtime analyzer

The runtime analyzer drives the diagnosis process by performing the following steps: 1) select a test to be executed; 2) interact with either the test engineer or the automatic test equipment to execute the selected test and collect its outcome; 3) based on the current partial syndrome, i.e., the outcome of all the already executed tests, identify the surely not faulty components and refine the set of the possibly faulty components; and 4) based on the specified stop policy, determine whether the diagnosis process should be halted or if it is necessary to repeat and continue from step 1.

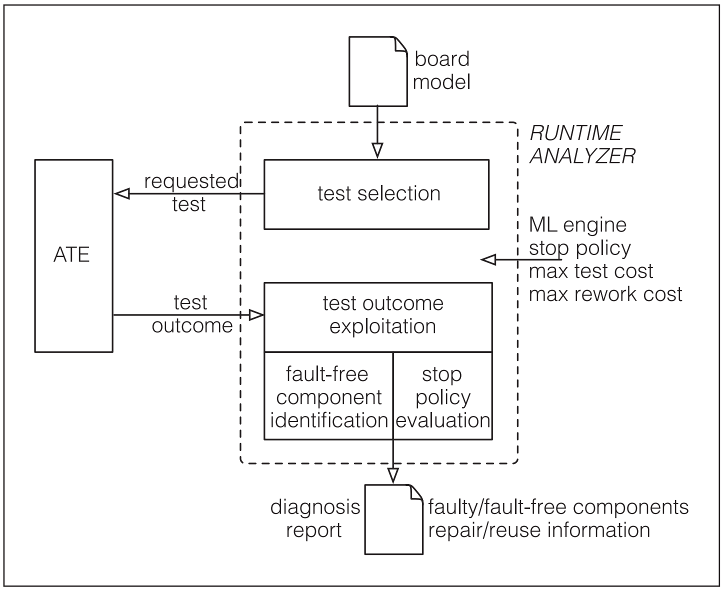

Figure 4. Architecture of the runtime analyzer.

The structure of the runtime analyzer is depicted in Figure 4: it takes as input the CTM model and other options, among which are the ML-based engine the designer wants to exploit and the desired stop policy. The tool is mainly composed of two subengines: the test selection engine and the test outcome exploitation engine. Their behavior is related to the adopted ML engine/policy; here we provide an overview of the framework, without delving in the specific characteristics, that have been comparatively investigated in [11].

The test selection engine determines the next test to be executed, according to the current partial syndrome. The test outcome exploitation engine is in charge of exploiting the current partial syndrome to determine: 1) which components can be considered as surely not faulty; 2) which components can be considered as faulty candidates; and 3) when to stop the diagnosis process according to the adopted stop policy. At present, the detailed information of *a priori* components' failure rates is not exploited in the framework. Thus, all components are considered to have an equal probability of being faulty. Varying probabilities will be considered in future work.

Two stop policies have been defined: the accuracy-aware and the innovative cost-aware policy. The former has the unique goal of minimizing the number of faulty candidates by excluding as many false positives as possible. The latter aims at finding a tradeoff between the achieved accuracy and the diagnosis cost, both in terms of test cost and board fixing cost (rework cost plus components cost). In more detail, when using the new cost-aware policy, the test/diagnosis engineer has to specify two parameters, namely $C_{\max_t}$ and $C_{\max_f}$, as well as a cost value for all the tests and components in the CTM and the board rework cost. $C_{\max_t}$ represents the maximum affordable test cost; similarly, $C_{\max_f}$ represents the maximum affordable fixing cost. The engine keeps track of the sum of the costs of the already executed tests (C_t), as well as of the current fixing cost (C_f), i.e., the sum of the costs of the current faulty candidates and the board rework cost. When $C_t > C_{\max_t}$ or $C_f < C_{\max_f}$, the diagnosis process ends. This may occur before one or more faulty candidates have been identified; in such a case, all components not classified as fault-free are considered as faulty candidates. The rationale behind the cost-aware stop policy is twofold: it reduces the number of executed tests, and thus the tests cost (constraints on the test cost), and it allows one to avoid to excessively refining the set of faulty candidates when cost-wise there are no benefits (constraints on the components cost).

Benefits

Here, we report some results from the application of the proposed framework to three industrial boards, whose characteristics (e.g., number of components n_C, tests n_T, and legal syndromes n_{LS}) are shown in the left-hand part of Table 1.

We first applied the static analyzer; the number of not isolated components (#NICs), of rarely

Table 1 Industrial boards: Characteristics and static analyzer results.

ID	n_C	n_T	n_{LS}	#NICs	#RICs	#RUTs
Board 1	5	18	11788	0	2	10
Board 2	14	24	12290	8	4	11
Board 3	25	44	13205	12	9	33

isolated components (#RICs) and of rarely used tests (#RUTs) are reported in the right-hand part of Table 1. It is worth noting that the difficult-to-diagnose components (#NICs plus #RICs) represent a high percentage of the total number of components (about 69% on average). Similarly, the percentage of rarely used tests is high (about 59% on average).

The runtime analyzer has then been used to evaluate the benefits of the incremental approach, with and without cost constraints, with respect to the “traditional diagnosis solution” based on the

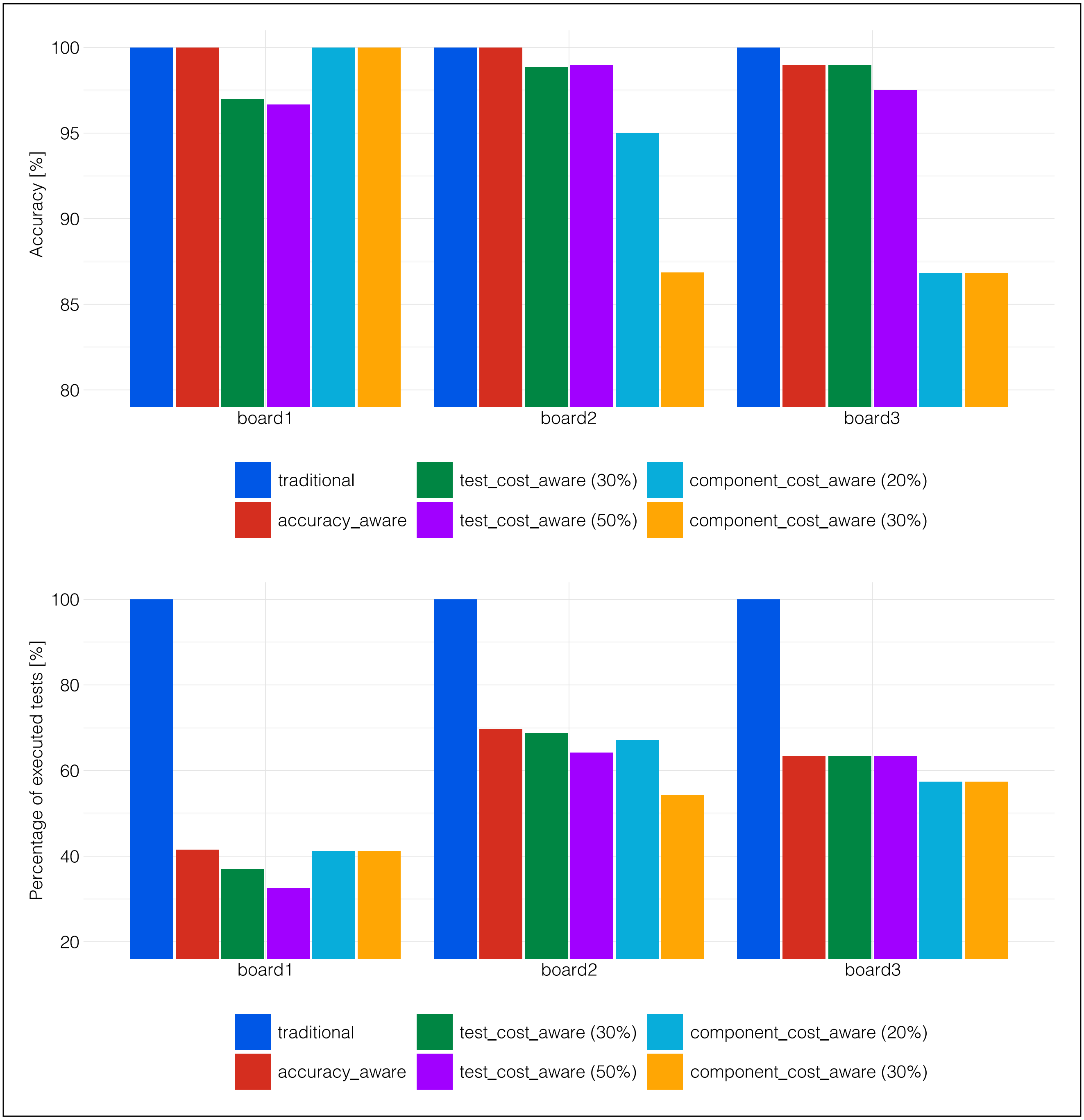

Figure 5. Accuracy and executed tests achieved by the traditional diagnosis and by the proposed framework.

execution of the complete test suite. We evaluated and reported the average accuracy of the diagnosis and the number of executed tests to identify the faulty candidate(s). We used the DT-based diagnosis engine because, as shown in [11], it seems to be the most scalable and robust engine, given the size of the analyzed boards. For each board, three campaigns have been executed

1) accuracy-aware policy: minimization of the number of tests, provided a complete accuracy is achieved;
2) test cost-aware policy, with a constraint on the test cost, such that the diagnosis is interrupted when the cost of the executed tests reaches a fixed percentage of the cost of the complete test suite;
3) component cost-aware policy, with a constraint on the components cost, such that the diagnosis is interrupted when the cost of the components identified as faulty decreases below a fixed threshold.

For cost-aware policies, when test costs are adopted as the main constraint, we ran two experiments using a 30% and 50% threshold, interrupting the diagnosis when 30% and 50% of tests remain, respectively. In this case, the configurable framework allows one to decide how much effort to use, in relation also with the use of the test machinery and time.

When component costs are used to constrain the diagnosis, the adopted thresholds are 20% and 30%, so that the diagnosis is interrupted when the cost of the presumably faulty components is 20% and 30% of the board total components' cost, respectively. In this case, there are always several false positives, thus lowering the level of accuracy; however, since the total cost of the faulty components erroneously considered as faulty is acceptable, this is not an issue. This solution is preferable when the user deems as not interesting to exactly know whether a component is faulty or fault-free if its substitution incurs a limited cost/rework effort.

Figure 5 reports the comparison of the accuracy and the number of executed tests achieved by the traditional diagnosis procedure and by the proposed framework with the different policies. The proposed framework always achieves a high accuracy (about 96% on average) and largely reduces the percentage of executed tests (about 55% on average). In general, although compromising accuracy, the approach minimizes diagnosis costs, while meeting the user's constraints.

The proposed cost-aware diagnosis framework implements a fully automated incremental adaptive approach to identify faulty components, allowing the diagnosis engineer to configure the process to meet his/her constraints and priorities that may differ in different contexts. Therefore, a unique characteristic of this framework is the possibility of configuring it to leverage accuracy for tests/components cost-related benefits.

Acknowledgements

This work was supported in part by the Cisco University Research Program Fund—Gift #2014-130689 (3696), an advised fund of Silicon Valley Community Foundation.

References

[1] P. Maxwell, I. Hartanto, and L. Bentz, "Comparing functional and structural tests," in *Proc. Int. Test Conf.*, 2000, pp. 400–407.

[2] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, "Adaptive board-level functional fault diagnosis using decision trees," in *Proc. Asian Test Symp.*, 2012, pp. 202–207.

[3] F. Hongxia, K. Chakrabarty, W. Zhiyuan and G. Xinli, "Diagnosis of Board-Level Functional Failures Under Uncertainty Using Dempster–Shafer Theory," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 31, no. 10, pp. 1586–1599, 2012.

[4] F. Hongxia, K. Chakrabarty, W. Zhiyuan and G. Xinli, "Board-level functional fault diagnosis using artificial neural networks, support-vector machines, and weighted-majority voting," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 32, no. 5, pp. 723–736, 2013.

[5] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty, "Board-level fault diagnosis using bayesian inference," in *Proc. VLSI Test Symp.*, 2010, pp. 244–249.

[6] F. Ye, K. Chakrabarty, Z. Zhang, and X. Gu, "Information-theoretic framework for evaluating and guiding board-level functional-fault diagnosis," *IEEE Design Test*, vol. 31, no. 3, pp. 65–75, 2014.

[7] L. Amati et al., "An incremental approach to functional diagnosis," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2009, pp. 392–400.

[8] Fault Detective, 4.0, Agilent Technologies. [Online]. Available: www.agilent.com/find/fd

[9] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu, “Board-level functional fault diagnosis using multikernel support vector machines and incremental learning,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 33, no. 2, pp. 279–290, 2014.

[10] C. Bolchini, L. Cassano, P. Garza, E. Quintarelli, and F. Salice, “An expert cad flow for incremental functional diagnosis of complex electronic boards,” *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 34, no. 5, pp. 835–848, 2015.

[11] C. Bolchini and L. Cassano, “Machine learning-based techniques for board-level incremental functional diagnosis: a comparative analysis,” in *Proc. Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst.*, 2014, pp. 246–251.

[12] C. Bolchini and L. Cassano, “A novel approach to incremental functional diagnosis for complex electronic boards,” *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 42–52, 2016.

Cristiana Bolchini is a Professor at Politecnico di Milano, Milan, Italy. Her research interests cover the areas of design and analysis of digital system with a specific focus on dependability and context-aware data design and management. She has authored more than 100 papers. Bolchini has a PhD in automation and computing engineering from Politecnico di Milano.

Luca Cassano is currently a Postdoctoral Research Fellow at Politecnico di Milano, Milan, Italy. His research focuses on the use of formal methods and machine learning techniques for fault simulation, test-ing, untestability analysis, diagnosis, and verification of digital circuits/systems. Cassano has a PhD in informa-tion engineering from the University of Pisa, Pisa, Italy.

Direct questions and comments about this article to Cristiana Bolchini, Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milan, Italy; cristiana.bolchini@polimi.it.