Systematic Software Testing of Critical Embedded Digital Devices in Nuclear Power Applications

Athira Varma Jayakumar[1], Smitha Gautham[1], Richard Kuhn[2], Brandon Simons[1], Aidan Collins[1], Thomas Dirsch[3], Raghu Kacker[2], and Carl Elks[1]

[1] *Virginia Commonwealth University, Richmond VA, USA*

[2] *National Institute of Standards and Technology Gaithersburg, MD, USA*

[3] *Razorcat, Inc. Berlin, Germany*

***Abstract* – While design assurance and testing methods for safety-critical systems have been widely researched and studied for years across a number of industry domains, there are few efforts reported in the literature on the *actual application* of software testing methods to nuclear power digital I&C systems or devices. We see this as a gap in the knowledge basis. The motivation for this research was to investigate the efficacy and challenges that arise when planning, automating and conducting systematic software testing on actual real-time embedded digital devices. In this paper, we present results on the application of a systematic testing methodology called Pseudo-Exhaustive testing. The systematic testing methods were applied at the unit, module integration levels of the software. The findings suggest that Pseudo Exhaustive testing supported by automated testing technology is an effective approach to testing real-time embedded digital devices in critical nuclear applications.**

Keywords - t-way combinatoric testing, Pseudo Exhaustive testing, software testing, safety critical systems, embedded digital devices

I. Introduction and Background

Nuclear power generation facilities worldwide are steadily trending toward "aging infrastructure" with the average age of a Light Water Reactor in the United States at about 37 years. To address this concern, global energy organizations and national energy agencies like the International Atomic Energy Agency and the US Department of Energy have introduced research and development programs to assist utilities in what is known as "plant modernization and upgrades". These modernization efforts will rely heavily on computer-based or digital Instrumentation and Control (I&C) systems for the replacement of obsolete analog or rudimentary digital systems. Even though advanced digital I&C devices have been used extensively in many other industries, their use in the nuclear industry for safety-related functions are still limited.

The U.S. Nuclear Regulatory Commission (NRC) identifies two design methods that are acceptable for eliminating Software Common Cause Failure (SCCF) concerns: (1) diversity or (2) testability (specifically, 100% testability) [1]. There is near universal consensus among computer scientists and software test engineers that enumerated exhaustive testing for modestly complex devices or software is infeasible [2]. For this reason, diversity and defense-in-depth architectural methods have become the norm in the nuclear industry for addressing vulnerabilities associated with SCCFs [1]. However, the disadvantages of large scale diversity methods for plant modernization are significant implementation costs, and increased plant integration complexity. Consequently, there has been much interest within the nuclear industry in the past 10 years toward finding cost effective testing methods for design assurance that go beyond defense-in-depth and diversity.

The pragmatic motivation for this study was to investigate the efficacy and challenges that arise when planning, automating, and conducting systematic software testing on actual real-time embedded digital device, as little has been published with respect to systematic software testing with respect to nuclear digital devices. In this paper we present results on the application of a systematic testing methodology called "Pseudo-Exhaustive" testing built around t-way combinatorial testing, partitioning, boundary value analysis, and path analysis [3]. Our selection of Pseudo Exhaustive testing methods was influenced by several important factors. First, the nuclear regulatory guidelines of [1] emphasize the testability of software, with a goal of testing all possible inputs and paths, which is intractable for current digital systems. Second, the principle elements of Pseudo Exhaustive testing have an established technical basis with a history of use in embedded safety critical systems [4]. In our work, the testing methods were applied at the unit, module integration levels of the software of an embedded smart sensor. We provide findings on the application of this systematic testing that helps inform the nuclear power community and other safety critical communities on broader impacts.

II. Testing Methodology and Study Design

Our approach builds on a test method called "Pseudo Exhaustive" [5] [6]. The definition of Pseudo Exhaustive is given below.

Definition: *Pseudo Exhaustive* - Software testing is considered bounded or Pseudo exhaustive when well-formed relations between input space and state space allow the testable state space to be reduced – enabling a feasible testable set. The key assumption is that the state space reduction process must preserve the properties of and among the elements from the original state space.

Our realization of Pseudo Exhaustive testing combines a number of testing techniques to satisfy the above definition. Specifically, our test methodology is organized around the following well-accepted methods: Equivalence partitioning, Boundary Value analysis, t-way combinatorial tests and Path analysis (MC/DC criteria).

In order to construct a well-formed input space model we employ equivalence partitioning and Boundary Value Analysis (BVA) sampling methods to pre-analyze the software prior to generating t-way tests. Next, t-way "covering" test suites are generated from the reduced input model. Research conducted by National Institute of Standards and Technology (NIST) [5] on software failures for diverse application domains (e.g. aerospace, medical, finance, IT, etc.) strongly indicate that a majority of software failures are induced by interaction faults arising from the interaction of just a few parameters, mostly by two and three levels of interaction. In our study, experimentation is performed by varying the interaction strength t of the combinatorial testing and by varying the set of representative values v for an input partition space. The number of t-way combinatorial tests that is necessary has been shown to be proportional to [7].

$$number\ of\ tests\ \triangleq v^t \log n \qquad (1)$$

where v is the value span of the input variables or parameter (n), and n is the number of inputs parameters, and t is the number of interactions between parameters. The outcome evaluation from these tests are conducted using an oracle or by utilizing assertions implanted in the code. The outcomes from these experiments include the Pass/Fail results of the tests, various coverage metrics, and the identification of flaws.

A. Combinatorial Testing via Coverage Analysis

One of the underlying assumptions for effective t-way combinatorial testing is developing an input model that is comprehensive in reaching all regions of the decision logic on the software. One way to achieve this is by *coverage path analysis*. Coverage path analysis involves measuring the structural aspects of the program using several metrics like branch coverage, statement coverage, Modified Condition/Decision Coverage (MC/DC) coverage. Our study process aims at achieving >95% MC/DC coverage. Previous works have studied the relationship between combinatorial coverage and structural coverage with respect to the completeness of test suites [8].

Branch coverage condition: A test set provides 100% branch coverage for t-way conditionals if $M_t + B_t > 1$, where M_t = minimum combinatorial coverage at level t, and B_t=minimum proportion of t-way combinations that is guaranteed to trigger a branch within the code, where all variables in decision predicates have values from the variable set with minimum coverage characteristic M_t. [8]

On the basis of this theorem, it can be asserted that test suites based on covering arrays which have a minimum combinatorial coverage of 100%, would always result in branch coverage of 100% if the input model is sufficient and complete. Input model deficiencies are thereby caught through branch coverage analysis. Since we employ MC/DC coverage analysis which subsumes branch coverage, therefore our approach is sound. Finally, the systematic testing of the software architecture follows the well-known unit test, integration test, and system test paradigm [9]. Safety standards such as IEC 61508, ISO 62626, and DO-178B and so on highly recommend this type of systematic testing for high levels of design assurance.

III. REPRESENTATIVE EMBEDDED DEVICE – THE VCU SMART SENSOR

The Embedded Digital Device we use for this study is a Smart Sensor, which is representative of the type used in nuclear power applications [3 appendix B]. The hardware platform uses the ARM Cortex-M4 processor featuring 2MB of Flash Memory. The

application software is hosted by the ChibiOS real-time operating system to ensure the proper scheduling and execution of all periodic tasks within the system, which are handled by a priority-based scheduler.

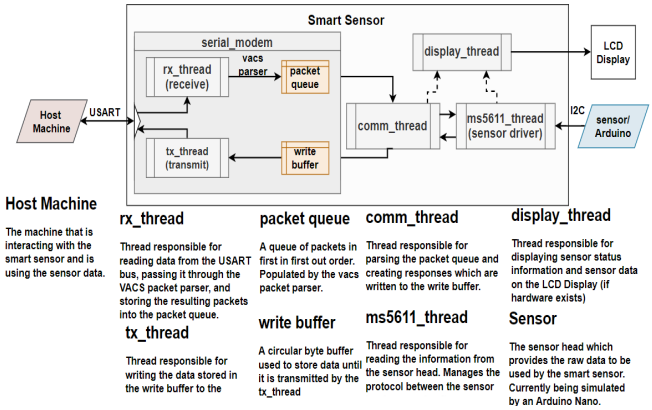

Figure 1: Smart Sensor Program Data Flow

Error! Reference source not found.1 shows the program data flow of the software components of the VCU Smart Sensor in its testing environment context, including the threads and communication protocols used to transmit data between modules. One of the critical threads is ms5611_thread, which is responsible for reading sensor head data and processing it. During the first phase of our research, we focused our testing strategy on the MS5611 thread.

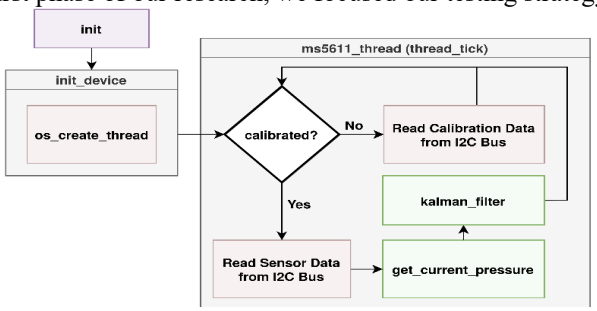

Figure 2: Basic flow of the ms5611_thread functions.

The basic flow of the ms5611 application thread is shown in Figure 22. The get_current_pressure function takes the raw temperature and pressure values from the sensor in the form of 32-bit unsigned integer values as inputs. The function processes the raw values and translates into pressure value, in Pascals. The kalman_filter function performs the Kalman Filter calculations in order to reduce noise. A floating-point value for the measured pressure values, prior to kalman filter calculations is used as the input, while the output is a floating-point value for the pressure values, following the kalman filter calculations. For both the functions get_current_pressure and ms5611_kalman_filter, test oracles were created to verify that they function as expected.

IV. TESTBED ARCHITECTURE

Our goal was to apply our test methodology on real hardware operating at real-time speeds. Our test bed architecture in Fig. 3 shows the basic workflow, the tools, hardware components and test artifacts necessary to conduct the real-time systematic testing experiments on the smart sensor. The three major tools used in our testbed are (1) Razorcat's TESSY, (2) National Institute of Standards and Technology (NIST) ACTS tool, and (3) Keil Interactive Debugger. TESSY, developed by Razorcat is an automated testing tool for safety critical embedded systems software [10]. TESSY provides test automation management for unit testing, integration testing directly on real-time embedded hardware. The ACTS tools from US NIST provide capabilities to automatically generate effective t-way combinatorial test cases. Additionally, ACTS has features to support for BVA, event sequence testing, covering array generation.

The Keil interactive debugger software is the interface between TESSY software and the target hardware. TESSY exploits the ARM Serial Wire Debugger (SWD) port which enables test vectors to be directly executed on target HW, and offers an unobtrusive extraction and monitoring of program data that includes I/O data, local and global variables, state variables, conditionals and guards. Referring to Fig. 3, the necessary C/C++ source files are loaded into the TESSY tool. TESSY analyzes the source files and populates all the local functions, external functions, external variables, global variables and macros. TESSY's classification tree editor facilitates BVA and equivalence partitioning, to confine the input domain into a tractable

set of values. The Software requirements, software design documents and the datatypes of the variables identified by Tessy during source code analysis guide the testers in creating various classes to represent equivalence partitions of the input domain and sub classes to feed in the boundary values for each partition. These representative test input data values fed into the classification tree are then exported from TESSY into excel and xml files.

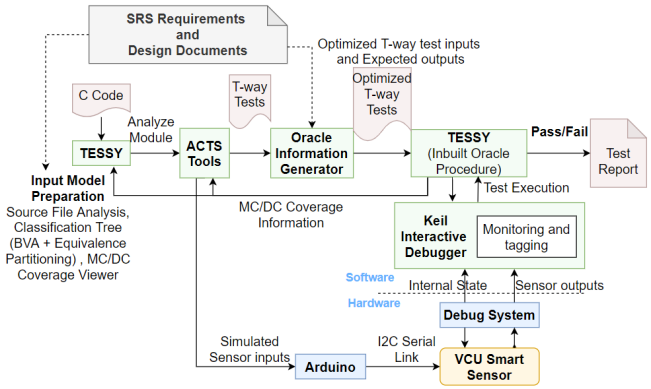

Figure 3: Baseline testbed architecture [3]

A custom script parses the Tessy exported xml file, which contains all the test data values, and converts it into input language that the ACTS tool can recognize. After importing the input model information that contains the input parameters (n) and their values (v), and further feeding in the required interaction level (t = 2 to 6) and parameter constraints if any into ACTS, the tool is run to produce t-way tests that contain all t-way combinations of input values that are specified in the classification tree in TESSY. The t-way test cases exported from ACTs are translated back into TESSY import format by a translation script. To address the oracle issue, algorithm/equations can be inserted into TESSY Epilogue and Prologue function which allows for automatic 'expected output' calculation during runtime. These functions act like a "shell" for automating the comparison of oracle to actual results. To support diversity in our oracle module we synthesized a kalman filter from MATLAB and Simulink and auto-generated the code from those models. TESSY runs test cases directly on the actual Smart sensor target by interfacing with the Keil debugger. The real time values of program data from the software are collected with the help of TESSY's inbuilt timestamp mechanism and compared with the fed in oracle data. The final step in the workflow after the initial test execution is the coverage analysis. MC/DC coverage data is generated in TESSY. This coverage data information is used to enhance tests to improve the code coverage as shown in Fig. 4. Low coverage due to uncovered branches involving more than t variables, can be improved by generating a higher t-way interaction test vectors in ACTs. In some cases, an incomplete input model could be the reason for low coverage - for example, due to essential parameters or necessary values for testing being not considered during input model creation.

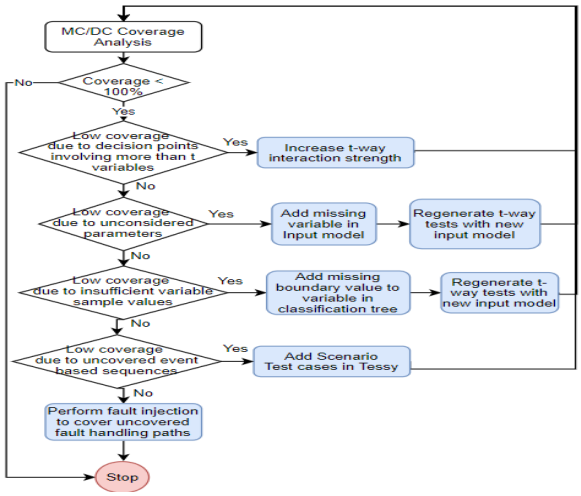

Figure 4: Coverage Analysis and Optimization process

V. RESULTS OF PRELIMINARY TESTING

Our testing on thread functions starts with the baseline test of 2-way combinatorial testing which then proceeds to 3, 4, 5 and 6-way tests. The main objective of the test experiments was to identify the level of *t* at which 100% of the interaction faults in the embedded software are detected cumulatively. Our experiments revealed that all the defects identified in the smart sensor software were found at the 2-way combinatorial testing level. This is not a general result, rather specific to the nature of the logic interactions in the smart sensor software. However, it is confirmatory evidence to other findings that suggest most faults are detected at lower interaction levels. The second aim was to analyze the impact of combinatorial testing on path coverage and thereby fault detection capability. On the thread functions that we conducted our experiments, a high 100% branch coverage was obtained at the baseline 2-way testing level itself [3].

Error! Reference source not found. presents the summary of results for unit testing on two functions; (1) get_current_pressure, and (2) Kalman_filter within the ms5611_thread in the smart sensor software. The number of test vectors rose exponentially with increase in t, as anticipated based on the relationship formula (1). The number of values sampled for each variable varies between variables and depends on their respective input domain and their usage in the code. The average value of variable values was around 7 values/variable. From the data collected, it can be seen that on average it took 400ms to execute and evaluate a single test directly on the target hardware. The achievable test productivity with this methodology was about 210,000 test vectors/day which is noteworthy when considering actual testing on a physical HW platform. The combinatorial unit tests were successful enough to uncover three native bugs (as shown in TABLE 2), in the smart sensor software that had been in use for years.

Function	T	Testing Type	Testing Method	Coverage	# Tests	Time	Pass/Fail	# Defects
get_current_pressure	T=2	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	21	< 1 min	21/0	1
	T=3	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	76	1 min	76/0	1
	T=4	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	285	2.3 min	285/0	1
	T=5	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	870	8 min	870/0	1
	T=6	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	2411	15 min	2411/0	1
kalman_filter	T=2	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	48	< 1 min	42/6	2
	T=3	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	316	3 min	276/40	2
	T=4	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	1608	12.4 min	1389/219	2
	T=5	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	7776	59.1 mins	5855/1921	2
					9748			

Another interesting bug caught by the combinatorial tests, which was not expected, is a potential buffer overflow vulnerability that existed in the get_current_pressure code. Such an error could be triggered by a hardware fault or exploited by a cyber-attack. We found test cases with raw pressure value greater than the maximum 24 bits (3 bytes) value '16777215' was expected to fail. But they ended up with 'Passed' results because the input buffer was allowed to overflow.

Few test cases resulted in 'Infinite' value test outcomes when the kalman filter computations ended up in 'division by zero'. The 'infinity' output value is an undesirable outcome and revealed that the code was missing a check to ensure that the divisor is not zero before performing division. Another issue identified was the missing overflow checks during float computations which resulted in 'Not a Number' outcomes for few testcases. 'NaN' (Not a Number) is an undefined result obtained when the processor attempts a 0/0 or Inf/Inf computation. A check for an overflow during computations and saturating the result to the maximum value of the float datatype 3.40282E+38 could prevent the computation result from ending up in infinity thereby avoiding infinity and undefined final outputs. These severe bugs seem to have appeared due to the deficiency of robust coding practices.

TABLE 1: SUMMARY OF RESULTS[3]

Function	T	Testing Type	Testing Method	Coverage	# Tests	Time	Pass/Fail	# Defects
get_current_pressure	T=2	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	21	< 1 min	21/0	1
	T=3	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	76	1 min	76/0	1
	T=4	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	285	2.3 min	285/0	1
	T=5	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	870	8 min	870/0	1
	T=6	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	2411	15 min	2411/0	1
	T=2	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	48	< 1 min	42/6	2

kalman_filter	T=3	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	316	3 min	276/40	2
	T=4	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	1608	12.4 min	1389/219	2
	T=5	Unit	BVA, T-way, MC/DC	Statement=100% MC/DC=100%	7776	59.1 mins	5855/1921	2
					9748			

Another interesting bug caught by the combinatorial tests, which was not expected, is a potential buffer overflow vulnerability that existed in the get_current_pressure code. Such an error could be triggered by a hardware fault or exploited by a cyber-attack. We found test cases with raw pressure value greater than the maximum 24 bits (3 bytes) value '16777215' was expected to fail. But they ended up with 'Passed' results because the input buffer was allowed to overflow.

TABLE 2 SOFTWARE ISSUES IDENTIFIED BY TESTING [3]

Issues	Software Function	Caught by
Missing Divide by Zero check	ms5611_kalman_filter	2-way combinatorial unit testing
Missing Overflow check in float computations	ms5611_kalman_filter	2-way combinatorial unit testing
Function processes input values outside valid range	ms5611_get_current_pressure	2-way combinatorial unit testing

In addition to verifying the handling of program data in threads/functions using combinatorial testing, the sequence of function calls (at the right time and right order) within a thread must be validated with robust test sets to determine if race conditions or disordering of events can occur. To verify event-based sequences and validate the correctness of the thread functions that have a number of external and local function calls driven by control flow logic, sequence-based tests were formulated in TESSY's Sequence editor. Sequence-based tests help to validate the temporal behavior of the indefinitely running threads scheduled by the real-time operating system (RTOS). Due to page limitations, we do not go into details of this testing we refer the reader to [3]. Testing of call graph execution sequence was more intricate and challenging as we had to interact with RTOS scheduler while it invoked tasks in the correct order and accurately emulate the function call order as per the call graphs. We found no faults in the sequencing of the function calls.

VI. CHALLENGES

The major challenges we faced are discussed below. One significant challenge we encountered was developing the testbed architecture to conduct the study. As the multi-pronged testing approach involved multiple tools and software (ACTS, Tessy, Keil uvision), we had to build an intricate workflow to interconnect them in order to ensure an efficient testing process. The challenge in testing real-time systems is that they need to be tested in the temporal domain as well as the value domain. Testing in the temporal domain implies the need to issue inputs at a precise time instance i.e. beginning of the scan cycle, controlling the state of the SW object at the start of the test execution and observing the timing of the result at the end of execution. These tests were complicated to develop as it required detailed knowledge of the control flow sequencing.

One other challenge we faced during our study is the identification of a reliable oracle to validate the correctness of the test results. Several approaches were explored. TESSYs option to feed in the oracle algorithm as user-specific C code, provides an automated dynamic oracle information generator. But this method could lose its efficiency when it comes to larger and more complex software. Another interesting approach was to use a diverse version of the software algorithm from which the reference values to verify the DUT can be derived. Simulink was used to create a kalman filter model and then embed the generated code as the oracle algorithm into TESSY.

Input model preparation consumed the most time in the entire test process as it is not an automated process. BVA and equivalence partitioning of inputs require a detailed knowledge about the usage of the input parameters within the code. We found pointer to structures, pointer to pointers and structure with pointer members being abundantly used in the code and being passed across functions. Such constructs can cause misinterpretations of the code and result in incomplete or incorrect input models to be prepared by the testers. This can further reduce the software coverage achieved with the tests.

VII. FINDINGS AND RECOMMENDATIONS

The overall goal of this research was to investigate pseudo-exhaustive testing as a suitable method for testing safety-critical embedded software in the context of nuclear power applications. As a broad indicator of the method, we discovered three native defects in the code that were latent for some time, and in one case the defect was there for years. In conducting this study, we compiled a number of findings, limitations, and recommendations which we feel are informative to the community. We summarize a few of these below.

Finding 1: *Confirmatory evidence to previous studies*. The results we obtained on interaction faults provide farther evidence that most faults are found at lower levels of interaction (level 2 or 3). These findings provide guideposts

on the strength of t-way interactions that should be considered when conducting testing.

Finding 2*: t-way and path coverage analysis are highly synergistic*. The combination of t-way and path analysis (MC/DC) is a compelling approach for reasoning about "completeness" - when to stop testing. Path coverage analysis provides direct feedback to the strength of the testing. With a given coverage goal, deficiencies in the input model and the t-way interaction strength can be found via path analysis.

Finding 3: *Conducting tests on actual HW platforms requires highly coordinated testbed and tool support*. Pseudo-exhaustive testing of the type we conducted has a lot of moving parts. As such, test automation tools for test generation and management of tests directly on real-time embedded hardware are necessary.

Finding 4: *Efficacy of the approach appears to be suitable for safety critical nuclear applications*. Although research is continuing, and results are preliminary, the evidence suggests that Pseudo exhaustive testing provides the type of rigor required for certification of safety critical SW within nuclear applications.

Limitations: *White Box testing and control-oriented software*. The testing methodology we present is a white box testing approach and it requires access to requirements, specifications, and source code documents by a testing entity or authority. In addition, it seems to work best for control-oriented software, i.e. SW that has computations organized around modes, configurations, and decision logic.

Recommendations (farther needs)

An interesting aspect of combinatorial test methods is that they allow tight integration with approaches to test oracle generation, including conventional model-based test generation, and some novel approaches as well. Instantiating a model with values from a covering array, then generating counterexamples from property claims makes it possible to produce matching inputs and expected outputs **Error! Reference source not found.**. Methods developed for input model validation can be used with covering arrays to partially automate oracle generation as well [11]. Similarly, methods and tools to facilitate automated assistance with respect to constructing the input testing model would be significant gain in overall testing productivity for complex embedded applications. This aspect of the work consumed about 50% of the effort.

VIII. Related Work

Recent work by Kuhn et al. [7] [5] [12] studied the effectiveness of Combinatorial Testing (CT) in a variety of application domains, from critical systems (aircraft collision avoidance TCAS to web browsers). Their research has consistently shown that about 20%–70% of software faults were triggered by single parameters, about 50%–95% of faults were triggered by two or fewer parameters, and about 15% were triggered by three or more parameters. Recent work by this group has considered how sequences can be tested via CT [13], comparing t-way CT testing with random testing, and methods for generating test cases and oracles [14].

Woods et al [15] propose and demonstrate using hierarchical mutation testing methods on smart sensor platform for nuclear power applications. This work focused on the efficacy and power of model-based testing to develop test vectors for detecting a variety of postulated design faults. Diao [16] describes the benefits and necessary features for automated software testing to be productive. Guerra et al [17] describe process safety justification approaches and specific techniques that can be used in the justification of a smart instruments' software.

Disclaimer: *Certain products may be identified in this document, but such identification does not imply recommendation by NIST, nor that the products identified are necessarily the best available for the purpose.*

References

[1] U. N. R. Commission, "Guidance for Evaluation of Diversity and Defense-in-Depth in Digital Computer-Based Instrumentation and Control Systems," *Branch Technical Position*, pp. 7–19, 2007.

[2] R. W. Butler and G. B. Finelli, "The infeasibility of quantifying the reliability of life-critical real-time software," *IEEE Transactions on Software Engineering*, vol. 19, no. 1, pp. 3–12, Jan. 1993.

[3] C. Elks, A. Jayakumar, A. Collins, R. Hite, T. Karles, C. Deloglos, B. Simmons, A. Tantawy, S. Gautham., "Preliminary Results of a Bounded Exhaustive Testing Study for Software in Embedded Digital Devices in Nuclear Power Applications," Idaho National Laboratory U.S. Department of Energy Office of Nuclear Energy report INL/EXT-19-55606, Sep. 2019.

[4] J. D. Hagar, T. L. Wissink, D. R. Kuhn, and R. N. Kacker, "Introducing Combinatorial Testing in a Large Organization," *Computer*, vol. 48, no. 4, pp. 64–72, Apr. 2015.

[5] D. Kuhn and V. Okum, "Pseudo-Exhaustive Testing for Software," in *2006 30th Annual IEEE/NASA Software Engineering Workshop*, Columbia, MD, USA, Apr. 2006, pp. 153–158.

[6] K. Sullivan, J. Yang, D. Coppit, S. Khurshid, and D. Jackson, "Software Assurance by Bounded Exhaustive Testing," p. 10.

[7] D. Richard Kuhn, Renee Bryce, Feng Duan, Laleh Sh. Ghandehari, Yu Lei, and Raghu N. Kacker, "Combinatorial Testing: Theory and Practice | Semantic Scholar," *Advances in computers, Elsevier*, vol. 99, pp. 1–66, 2015.

[8] R. Kuhn, R. N. Kacker, Y. Lei, and D. Simos, "Input Space Coverage Matters," *Computer*, vol. 53, no. 1, pp. 37–44, Jan. 2020.

[9] K. H. Pries and J. M. Quigley, *Testing Complex and Embedded Systems*. CRC Press, 2018.

[10] "TESSY - Test System - Razorcat Development GmbH." https://www.razorcat.com/en/product-tessy.html (accessed Feb. 29, 2020).

[11] D. R. Kuhn, R. N. Kacker, Y. Lei, and J. Torres-Jimenez, "Equivalence class verification and oracle-free testing using two-layer covering arrays," in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2015, pp. 1–4.

[12] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial Coverage Measurement Concepts and Applications," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, Luxembourg, Luxembourg, Mar. 2013, pp. 352–361.

[13] D. Ri. Kuhn, J. M. Higdon, J. F. Lawrence, R. N. Kacker, and Y. Lei, "Combinatorial Methods for Event Sequence Testing," in *Verification and Validation 2012 IEEE Fifth International Conference on Software Testing*, Apr. 2012, pp. 601–609.

[14] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei, "Combinatorial Testing and Random Test Generation," in *Introduction to Combinatorial Testing*, Chapman and Hall/CRC, 2016, pp. 202–215.

[15] Richard Wood, Carol Smidts, Carl Elks, and Brent Shumaker, "An Approach to Model-based Testing of Instrumentation with an Embedded Digital Device," presented at the American Nuclear Society 11th International Topical Meeting on Nuclear Plant Instrumentation, Control and Human–Machine Interface Technologies (NPIC & HMIT), Orlando, FL, 2019.

[16] Xiaoxu Diao, Manuel Rodriguez, Boyuan Li, and Carol Smidts, "Automated Software Testing.," in *Analytic Methods in Systems and Software Testing*, John Wiley & Sons, 2018, p. 373.

[17] Sofia Guerra, Peter Bishop, Robin Bloomfield, and Daniel Sheridan, "Assessment and Qualification of Smart Sensors," Las Vegas, Nevada, 2010, pp. 7–11.