Toward a Systematic and Safety Evidence Productive Verification Approach for Safety-Critical Systems

Aiman Gannous
Department of Computer Science
University of Denver
Denver, USA
aiman.gannous@du.edu

Anneliese Andrews
Department of Computer Science
University of Denver
Denver, USA
andrews@cs.du.edu

Barbara Gallina
IDT, MRTC
Mälardalen University
Västerås, Sweden
barbara.gallina@mdh.se

***Abstract*—In safety-critical systems, the verification and validation phase in the software development life cycle plays an important role in assuring safety. The artifacts' outputs of the verification and validation processes represent the evidence needed to show a satisfactory fulfillment of the safety requirements. Providing strong evidence to show that the requirements of the domain standards are met is the core of demonstrating safety standards compliance. In this paper, we propose a systematic approach for verifying safety-critical systems efficiently by integrating model-based testing, combinatorial testing, and safety analysis; this is all driven by providing safety assurance. The approach provides both testing and formal verification capabilities, and it is easy to implement into a tool for use in an industry setting. To show how our approach could contribute to safety standards compliance, we investigated it's capability to fulfill the safety requirements by analyzing and linking the data produced from the steps in the approach to a safety evidence taxonomy.**

***Keywords*-safety certification; safety assurance; standards compliance; testing safety-critical systems; model-based testing; combinatorial testing; safety analysis;**

I. Introduction

The number of safety-critical functions that are being controlled by software is increasing, and as a result, the demand for correctness and availability of the software is increasing too [1]. To commercialize a safety-critical system, the software product must comply with the corresponding domain standards. A proof of compliance to these standards must be prepared and documented by the system developers and delivered to the corresponding certification governing body [2]. Certification is about arguing that the system is reliable and safe based on evidences that have an appropriate degree of confidence [3].

An example of these standards is the DO-178C [4], the standard for developing and assuring avionics. DO-178C is an objective-based standard as it contains a set of objectives that the developers need to meet. According to [5], more than half of the DO-178C objectives are verification objectives. Verification of safety-critical systems (SCS) is an important part of the certification process; as a result, testing is a primary requirement for safety certification. However, each domain has its own set of standards and guidelines on how to perform the testing activities and what objectives need to be met.

Some other examples of such standards and relevant domains are: ISO 10218 [6] for robots in an industrial environment, and ISO 13482 [7] for robots for personal use, EN 50126 [8] the European standard for safety in the railway domain, ISO 26262 [9] for the automotive domain and last but not least, IEC 62304 [10] for the medical software domain.

To obtain successful certification approval of their systems, developers are required to follow these standards throughout the software development life cycle. In some domains, providing a safety case is also required. A safety case as described by Bloomfield and Bishop in [11] is a "documented body of evidence that provides a valid argument that the system is safe for a given application in a given environment".

For safety-critical systems, arguing compliance is costly. Therefore, the need for a cost-effective compliance arguing process is pressing. Most of the mentioned domain standards urge conducting the verification activities at each phase of the software development life cycle. The use of a verification methodology that produces the proper safety evidences may offer efficiency and effectiveness in the compliance arguing process. Therefore, it is crucial to choose the adequate verification methodology that fulfills as many objectives or requirements of the domain standards as possible.

Formal methods have been used successfully in some safety-critical domains as verification techniques to prove correctness. However, in some domains, testing is still used to meet the required level of confidence in the product [1]. Airbus and Dassault-Aviation had successfully used formal methods for system verification. They showed that it is practical and cost-effective. However, at the software level, system-functional safety testing still could not be replaced by formal methods

[12]. Since formal methods are proven to be cost-effective but can not replace testing completely, we believe that the cost-effective verification approach could be the one that combines both formal verification and testing.

In certification, gaps between the developers and the compliance auditors exist, especially regarding the verification and validation requirements. One of the challenges which contribute to these gaps is the ambiguity of the certification standards. Policies and certificate regulators do not usually explain in detail how the compliance arguing process should be executed and what types of evidence should be presented; rather, they just specify what should be executed [13]. In addition, due to new advances in technology being introduced and used, such as Artificial Intelligence and Machine Learning techniques, current domains' de facto standards do not yet include clear guidance on how to verify and certify the systems that use these technologies. We believe that assessing verification methodologies with respect to certification activities will help in bridging these gaps. This will also motivate the researchers to build their methodologies based on meeting the certification requirements.

In this paper, we introduce a systematic verification approach based on Model-based testing (MBT), Combinatorial testing (CT) and safety analysis for the purpose of safety-critical systems verification. The approach is presented in five tasks using SPEM 2.0 models. We aim to contribute to narrowing the gaps between the developers and the compliance auditors by showing an assessment of our methodology contribution in the certification process. We measure the methodology adequacy in the arguing compliance process based on what safety evidences it could provide from the safety evidence taxonomy adopted from Nair et al. [14]. This taxonomy is a result of a systematic literature review classifying the artifacts, tools and methodologies outputs that were considered as evidences for safety.

The rest of the paper is structured as follows: In Section II, we present background information on the related subjects and tools used in this research. In Section III, we introduce our verification approach. In Section IV, we show an evidence-based assessment of the proposed approach regarding the certification process. In Section V, we discuss our findings. In Section VI, we provide an overview of the related work, and finally, in Section VII, we present concluding remarks and future work.

II. Background

A. Testing Safety-critical systems

To ensure safety of a safety-critical system (SCS), safety analysis techniques are used to identify possible hazards such as Fault Tree Analysis (FTA), Preliminary Hazard List Analysis (PHLA), Failure Mode and Effects Analysis (FMEA) [15]. Then the identified hazards must be shown to be prevented or mitigated. Testing SCSs is different from other classes of systems, since testing SCSs should be performed with the presence of failures to cover testing for proper mitigation of undesired behavior in addition to the normal functioning in the absence of failures. In testing SCSs, Sánchez et al. [16], proposed a test case generation methodology based on faults. Their methodology uses Extended Finite State Machines (EFSM) as a model-based technique to build the behavior model and construct fault trees to identify the possible failures in system states. Nazier et al. [17] also used a model-based approach for testing SCS's but they used model checking for system safety verification. Gario et al. [18] proposed a model-based approach for testing SCS using Communicating Extended Finite State Machines (CEFSM) to build the behavioral model and integrate the model with the transformed fault trees to generate test paths using different graph coverage criteria. All the previous mentioned contributions had to perform a compatibility step in their approaches since they are integrating behavioral models with fault models. Andrews et al. [19] proposed a methodology for testing proper failure mitigation of SCSs. They used an applicability matrix to identify the possible points of failure in test paths generated from the behavioral models. From the mitigation requirements of the system under test (SUT), they modeled the mitigation behavior using EFSMs and integrated them successfully into the identified points of failure to generate failure mitigation tests. These mentioned contributions present novel approaches for testing SCSs, however, none of them consider safety certification issues nor do they provide an assessment regarding safety evidence production.

B. Model-based testing (MBT)

Model-based testing uses models such as finite state machines (FSM) to model system behavior and then generate tests from these models [20]. In MBT, usually five steps are performed as follows:

1. Build the model.
2. Define the test selection criteria.
3. Generate test paths from the models.
4. Generate concrete test cases from the test paths and
5. Execute the test cases and report the results.

In their literature survey, Dias-Neto et al. [21] classified MBT methods into five different classes. The classification is based on the representation of information from the software requirements. One of the tools that has been used successfully in formal modeling is FSM. FSMs basic formalism does not provide the capability of modeling all the aspects of software components behavior such as conditions and triggers. However, the extended versions EFSM and CEFSM can cover this drawback [22].

C. Combinatorial testing (CT)

Combinatorial testing techniques such as pair-wise coverage are crucial in producing the minimal efficient test suite size since exhaustive testing is too expensive, without compromising the effectiveness [23]. Combinatorial methods as defined by Grindal et al. [24] in their literature survey, are methods that select test cases by choosing values then combine them strategically. In the survey, they classified these methods into

two classes, deterministic and non-deterministic. An example of a deterministic combinatorial method is the In-parameter-order algorithm (IPO). The IPO algorithm could be used to generate a pair-wise coverage-based test suite by creating a test suite for a subset of the identified parameters then add more parameters incrementally one at a time until all parameters are covered [25].

D. Systems Process Engineering Meta-Model Specification Version 2.0 (SPEM 2.0)

SPEM 2.0 provides a meta modeling graphical language of the software engineering process standardized by Object Management Group (OMG). SPEM 2.0 offers process oriented modeling elements. A description of a subset of these elements that are used in this research to model our approach are illustrated next. See [26] for more details and more SPEM 2.0 modeling elements.

Task: defines the work being performed.
Role: identifies who holds responsibility to perform the task.
Work Product: an element that is required as input to a specific task/s or an output that is produced by a task.
Guidance: identifies the source that provide the required information or knowledge to perform a task.
Fig. 1 shows some of the SPEM 2.0 graphical modeling elements.

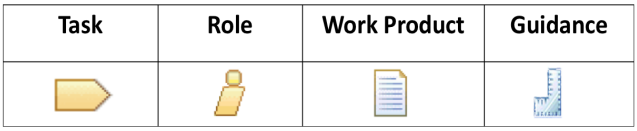

Fig. 1. **Sample of SPEM 2.0 modeling elements**

III. Approach: Model-Combinatorial based testing (MCbt)

Here we introduce our systematic testing methodology. The proposed methodology combines model-based testing, combinatorial-based testing and safety analysis techniques, aiming at providing various testing and verification activities that produce different safety evidences to support a successful and efficient safety certification process. The MCbt consists of five tasks, each task is illustrated in this paper using SPEM 2.0 meta modeling in Figures 2 - 6.

Task-1: The purpose of this task is to construct a behavioral model for each component in the system. As shown in Fig. 2, the tester creates models that are independent from the development team to perform testing activities. The inputs to this task are system requirements and safety specification. Here we recommend the use of formal modeling to allow the application of formal verification. The output is a model for each of the system under test (SUT) components.

Task-2: The purpose of this task is to generate test paths for each of the system component models. As illustrated in Fig. 3, the inputs to this task will be the behavior models created in task 1. The use of formal modeling techniques such as FSMs to model the system behavior will provide the capability of using various graph coverage criteria such as prime-path, edge, and node coverage criterion to automate the construction of these test paths [27]. The output of this task is a set of test paths for each component of the SUT.

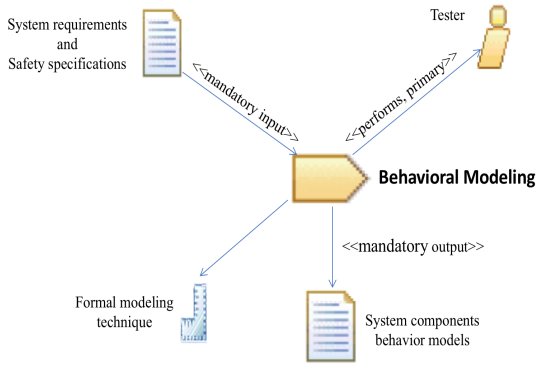

Fig. 2. **Task1, Building the behavioral models process**

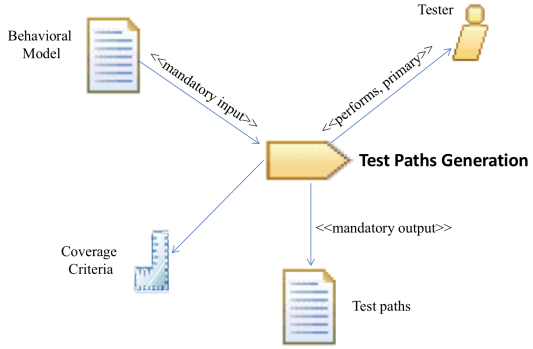

Fig. 3. **Task2, test paths generation process**

Task-3: As illustrated in Fig. 4, we use combinatorial techniques to combine test paths for different SUT components efficiently. In this task, the inputs are sets of test paths for two or more different components of the SUT. We could face a complexity challenge both in the time to create these combinations and in the size of the constructed combinations. However, these challenges can be overcome by using pair-wise combinatorial algorithms designed for combinatorial testing, such as In-parameter order algorithm [24]. Outputs of task 3 will be a combination of test paths for a set of SUT components.

Task-4: The purpose of this task is to perform a safety analysis to build a fault tree (FT) for each identified failure. Safety requirements of the SUT are used as inputs to this step to produce fault trees using fault tree analysis (FTA) as guidance. The output from task 4 will be a fault tree for each possible failure. This task is shown in Fig. 5.

Task-5: The purpose of this task is to classify the combination of test paths based on different testing targets. The inputs are the FTs that were produced from task 4, and a set

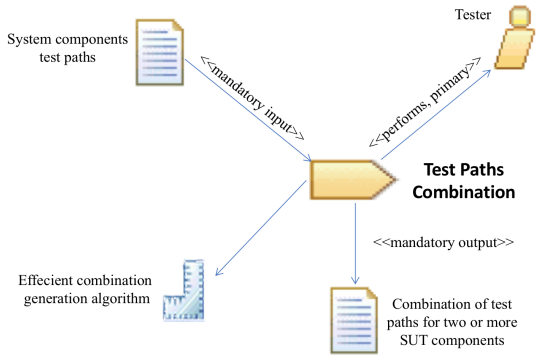

Fig. 4. **Task3, test paths combination process**

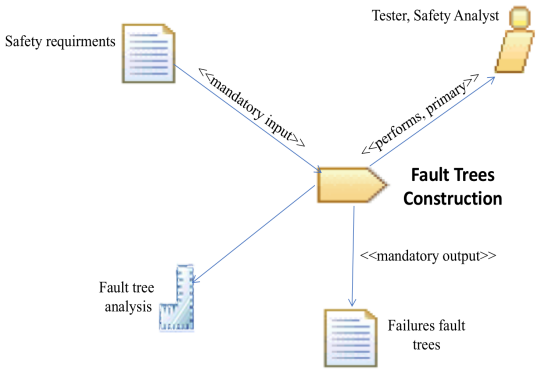

Fig. 5. **Task4, constructing failure fault trees process**

of combined test paths produced by task 3. The task is to use failure occurrence derived from the related FTs to classify the set of the inputted combined test paths. Two classes of test paths shall be identified; a subset of the combined test paths that does not produce failures, which will be used for normal behavior testing, and a subset of the combined test paths that create failures from undesired state combinations of different components. The later subset of test paths will be used for robustness and fail-safe testing of the SUT components. Task-5 process is shown in Fig. 6.

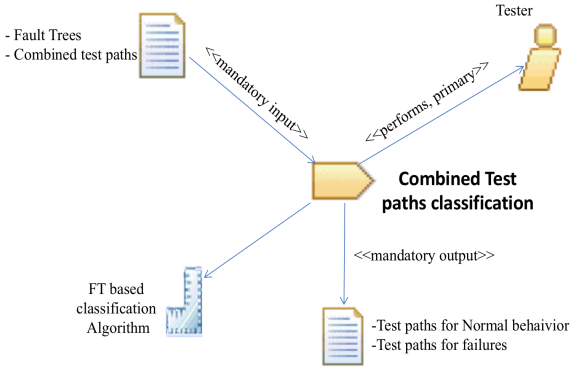

Fig. 6. **Task5, combined test paths classification process**

IV. Mapping MCbt outputs to Safety evidences

In this section we will assess MCbt's contribution to the certification process based on the safety evidence type it produces. The assessment metric will be based on the number of safety evidences it can provide. We do that by mapping MCbt outputs to the possible safety evidence type they can provide from the safety evidence taxonomy adopted from [14]. The taxonomy consists of 49 different evidence types, however, in the verification and validation (V&V) results category, we are interested in 16 types of safety evidence after excluding review and inspection activities evidences. In the safety analysis category, 5 types of safety evidence have been identified. Our evaluation of MCbt will be based on the possible mapping of MCbt outputs to these 21(16 + 5) evidence types as we propose a V&V approach that combines Model-based testing, Combinatorial testing and Safety analysis. To show the mapping, we will use the evidence type definition to match the output from MCbt steps.

The 21 evidence are categorized as follows: 5 from safety analysis, 3 from formal verification and 13 from testing.

In **Safety Analysis**, 2 out of 5 evidences are provided by MCbt. Those evidences are:

a. Risk analysis: defined in [28] as an analysis that provides expected danger specification in case of hazard activation.

b. Hazard cause specification: defined by [28] as the conditions and elements that enable a SCS to enter a hazardous state.

From the above definitions, safety analysis using fault trees is one of the techniques that could produce these two types of evidences. In MCbt, task 4 outputs (FTs) will provide these types of evidences. The rest of the evidence types in this category, *Accident specification*, *Hazard specification* and *Hazard mitigation specification* are not provided by MCbt at its current level of maturity. However, it is still probably possible to derive these evidences as the maturity level of MCbt increases or as more safety analysis techniques are added to the process.

In the **Formal verification** category, safety evidence usually can be derived from the results of three formal methods verification techniques: *Theorem proving*, *Model checking* and *Automated static analysis*. In MCbt, building the system component behavioral models is done in task 1. Using formal modeling techniques in this task such as CEFSMs or Timed Petri Nets, will enable the tester to use model checking. Results from Theorem proving, and Automated static analysis are not provided by MCbt as the first needs a non-graphical system representation, and the latter applies to source code which is not one of the inputs in the process.

In the third category of safety evidences, **Testing**, there are 13 evidences that could be obtained from the testing results, and they are classified into three subcategories as follows:

1. Target-based testing, three out of three types of testing in this subcategory are enabled using MCbt:

a. Unit testing: applicable using MCbt as in task 1, each component/unit behavior is modeled separately, and in task 2, test paths from each model will be generated using graph coverage criteria. These test paths are then used to create concrete test cases for each unit.

b. Integration testing: also applicable using MCbt in the collaboration of tasks 1, 2, and 3. As in task 2, test paths will be generated from the component models and in task 3, the combination of two or more components test paths will enable integration testing as a new component/unit behavior model is constructed.

c. System testing: on the system level, as a pair-wise combination is recommended for efficient combination of test paths from different components, a combination of all component test paths will provide the capability for system testing. Therefore, outputs from task 5's work product could be used for system testing.

2. Environment-based testing: only one type of testing in this subcategory is enabled using MCbt. It is not obvious how MCbt outputs could be used to contribute in *Non-operational testing*, however, in *Operational testing*, a pair-wise combination of all components test paths that is filtered using FTs as illustrated in task 5, could be used to verify the system behavior in its actual operating environment as Operational testing defined in [29].

3. Objective-based testing category: MCbt can produce evidences from the results of 4 out of 8 types of Objective-based testing. Those four testing types are:

a. Normal range testing: the verification of a critical system behavior under normal conditions as defined by [30].

b. Acceptance testing: the validation of a critical system behavior with respect to customer requirements as defined by [29].

c. Functional testing: as defined by [29], the validation of the conformance of the behavior of a critical system to its specifications.

From MCbt tasks, the component models are reflecting the normal behavior, the test paths originally generated from the components models, and the component models created using system specification and requirements, therefore, the above three types of testing can be performed using the work product output in task 5 of MCbt.

d. Robustness testing: as defined in [29], is the verification of a critical system behavior in the presence of failures.

This type of testing can be performed using the work product output in task 5 of MCbt. From the work product, the identified subset of combined test paths that leads to a failure will be used to test for robustness.

From the other four testing types in this subcategory, *Structured coverage testing* cannot be performed since MCbt is a black-box testing technique. Finally, based on their definitions in [29], *Performance, Stress and reliability* testing cannot be completely ruled out at this time. Their results could be implicit in MCbt tasks, and therefore, they need an experimental evaluation to see if they can be made explicit.

Fig. 7, adopted from [14], shows safety evidences related to V&V and safety analysis that are covered by MCbt. The green boxes represent the evidences that MCbt can provide, while the yellow ones are the evidences that need more experimental analysis to clarify MCbt's contribution toward them. The white boxes are the evidences that MCbt cannot provide.

Table. I shows MCbt SPEM 2.0 work product outputs that we successfully mapped to safety evidences in the safety evidence taxonomy.

TABLE I
SUMMARY OF MCBT OUTPUTS MAPPING TO SAFETY EVIDENCE TYPE

NO.	Safety Evidence Source	*Task*	*SPEM.WP*
1	Risk Analysis	4	Fault trees
2	Hazard Cause Specs.	4	Fault trees
3	Model Checking	1	Behavior Models
4	Unit Testing	2	Test Paths
5	Integration Testing	5	Normal/Failure Test Paths
6	System Testing	5	Normal Test Paths
7	Operational Testing	5	Normal Test Paths
8	Normal Range Testing	5	Normal Test Paths
9	Acceptance Testing	5	Normal Test Paths
10	Functional Testing	5	Normal Test Paths
11	Robustness Testing	5	Test Paths for failure

V. DISCUSSION

Our proposed verification approach, MCbt, was driven by safety evidence production for efficient safety-critical system verification and certification. The use of Model-based testing offers a systematic test path construction from system requirements, which provide a threefold advantage related to certification. First, it provides requirements-based testing activities, second, it provides traceability between test cases and high-level requirements, third, using models constructed independently from the developers design, provides independent testing. In addition, modeling the SUT components individually will allow us to perform unit testing and integration testing in parallel with the software development process.

Integrating combinatorial testing using pair-wise coverage algorithms, will keep the generation of test paths combination efficient while maintaining the required effectiveness.

Safety analysis using FTs will provide results from risk analysis and hazard cause specification, while at the same time being used as decision rules to identify test paths for failures from different SUT components' test paths combinations. Those identified test paths could be used for robustness and fail-safe testing.

TABLE II
SAFETY ANALYSIS AND VERIFICATION & VALIDATION RELATED EVIDENCE TYPES USED IN DIFFERENT SCS DOMAINS. X MEANS EVIDENCE HAS BEEN IDENTIFIED TO BE USED IN THE DOMAIN.

NO.	Evidence Source	*Aerospace*	*Automotive*	*Aviation*	*Medical*	*Maritime*	*Nuclear*	*Railway*	*Robotics*
1	Risk Analysis	x	x	x	x	x	x	x	x
2	Accident Specification	x	x	x	-	x	x	x	x
3	Hazard Mitigation Specification	x	x	x	x	x	x	x	x
4	Hazard Specification	x	x	x	x	x	x	x	x
5	Hazard Cause Specification	x	x	x	x	x	x	x	x
6	Automated Static Analysis	x	x	x	x	-	x	x	x
7	Model Checking	x	x	x	x	-	x	-	x
8	Theorem Proving	x	x	x	-	x	-	x	x
9	Operational Testing	x	-	x	-	-	-	-	x
10	Non-operational Testing	-	-	x	x	-	-	x	-
11	Normal Range Testing	-	x	-	-	-	-	x	x
12	Acceptance Testing	-	-	x	-	-	x	x	-
13	Functional Testing	-	-	x	x	-	-	-	-
14	Robustness Testing	-	x	x	x	-	x	-	-
15	Structural Coverage Testing	x	x	x	-	-	x	x	x
16	Performance Testing	x	-	x	-	-	-	-	-
17	Stress Testing	x	-	x	-	-	-	-	-
18	Reliability Testing	x	-	x	-	-	x	x	-
19	Unit Testing	x	x	x	x	x	x	x	x
20	Integration Testing	x	x	x	-	-	x	x	-
21	System Testing	x	-	x	-	-	-	-	-
	Total	**16**	**13**	**20**	**10**	**7**	**13**	**14**	**12**

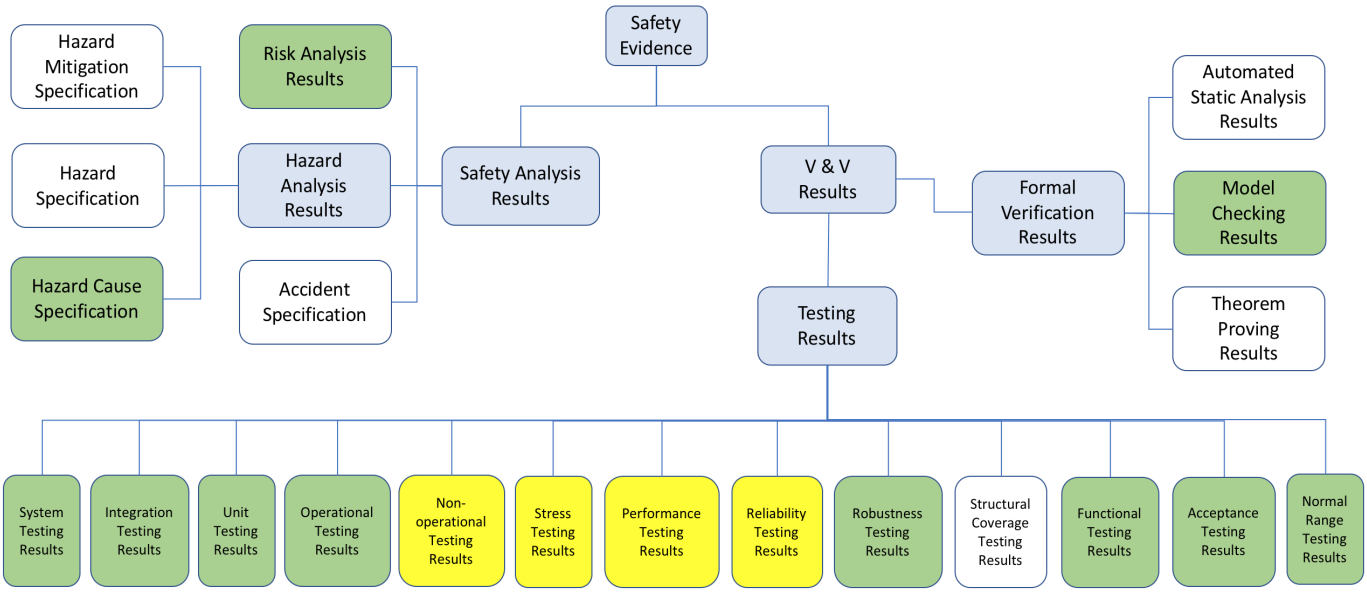

Fig. 7. **Safety evidences coverage by MCbt. Evidences provided by MCbt are shown in green filled boxes. Yellow filled boxes are evidences that need experimental analysis. Evidences in the white filled boxes are not provided by MCbt.**

Regarding safety evidences production, from 21 safety evidences divided between 3 classes, *safety analysis, formal verification and testing*, 11 safety evidences were provided by MCbt. As illustrated in Fig. 8, from the safety analysis class MCbt provides 2 out of the 5 (40%) safety evidences, while in V&V, MCbt could provide 1 out of the three (33%) formal verification activities and 8 out of 13 (61%) of the testing safety evidences.

Adopted from [14], in safety-critical domains, each one has a different set of evidence that are used in the certification process derived from V&V activities and safety analysis. Based on that, MCbt safety evidences production covers multiple domains with slight variations.

From Tables I and II, in the Aerospace domain, 16 different safety evidence out of the 21 V&V activities and safety analysis are recorded to be used in the certification process. MCbt provides 7 (44%) of them. In the Automotive domain 13 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 6 (46%) of them. In the Aviation domain, 20 different safety evidence out of 21 are

recorded to be used in the certification process. MCbt provides 10 (50%) of them. In the Medical domain, 10 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 5 (50%) of them. In the Maritime domain, 7 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 3 (43%) of them. In the Nuclear domain, 13 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 6 (46%) of them. In the Railway domain, 14 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 5 (36%) of them and in the Robotics domain, 12 different safety evidence out of 21 are recorded to be used in the certification process. MCbt provides 6 (50%)of them. Fig. 9, shows these percentages of the safety evidence that MCbt provides in each safety-critical domain. The higher evidence coverage percentages were in aviation, medical, and robotics domains with 50% of the recorded evidence provided, while the lowest was in the railway domain with 36% of the recorded evidence provided by MCbt. We can also observe that MCbt provides three types of evidence that have been identified in all SCS domains, Hazard Cause Specification, Risk Analysis Results and Unit Testing Results.

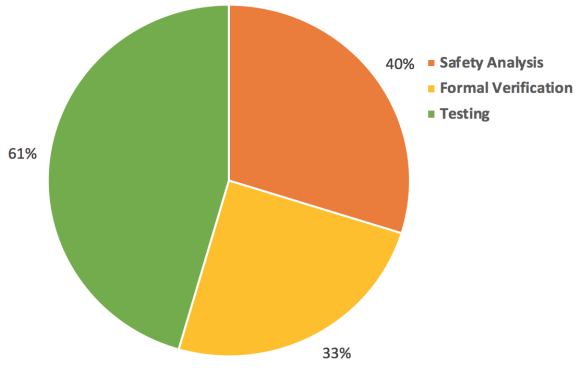

Fig. 8. **MCbt safety evidences production in each V&V activities type.**

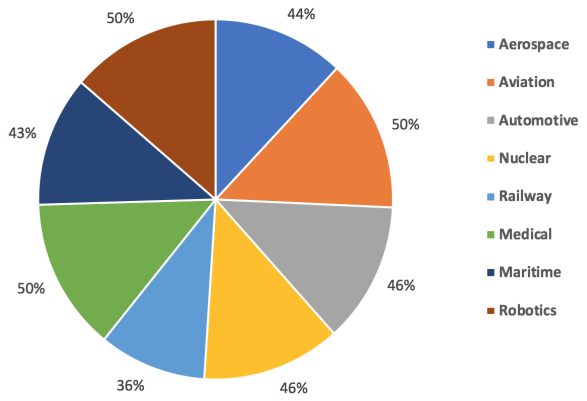

Fig. 9. **MCbt safety evidences use in common safety-critical domains.**

VI. Related work

Arguing whether a development or a verification technique is fulfilling the required objectives in a given activity by domain standards is an active research field that needs more practical evaluation by both academia and industry. Cârlan et al. [31] contribute to the problem of appropriately employing verification techniques in the avionic domain by extending the Structured Assurance Case Meta-model (SACM), describe relationship types between heterogeneous verification results collaborating to the achievement of one safety goal and use the proposed meta-model to provide safety case patterns for arguing the appropriateness of a certain technique.

Gannous et al. [2] introduced an end-to-end testing approach for testing safety-critical systems called Fail-safe Model Based Testing (Fail-safeMBT). Fail-safeMBT is the result of merging Gario et al. [18] approach and Andrews et al. [19] approach. The two approaches share a major activity as they both use CEFSM to build the behavior model for test generation. Gario et al. approach aims at generating failures, while Andrews et al. approach aims at verifying the mitigation for these failures. Gannous et al. applied Fail-safeMBT to the Autopilot system basic functionality as a case study. They also examined the adequacy of Fail-safeMBT in the standard compliance process by linking what it could achieve in DO-178C Verification of the Verification process results objectives compliance. They argued by using evidence produced from the process outputs and showed that Fail-safeMBT offers partial compliance with the standards, specifically as a requirements-based testing technique, providing complete coverage in testing high-level requirements in addition to providing the independence property when required by the safety level.

Also, in [32], Gallina et al. used a Model Driven Safety Certification method to show that Fail-SafeMBT, can be partly used as testing-planning related evidence within a safety case. They focused on specific sections of the DO-178C and its supplements related to verification process compliance and argued about Fail-SafeMBT compliance with the related verification process planning elements.

Cofer et al. [3], investigated the use of formal methods and tools in the certification of a Flight Guidance System design that is deployed in commercial aircraft. Three case studies were provided to show the use of theorem proving, model checking, and abstract interpretation in satisfying related certification objectives defined in DO-178C and its supplement, DO-333. They showed how each technique could be applied to different life cycle data items and how the evidence produced by these three techniques might be used in the certification process.

VII. Conclusion and Future work

The need for a better safety-critical system development process that produces clear safety evidences is one of the main challenges in safety-critical system certification [14]. Verification is not just an important phase in any software development process, but also tends to be included in every development life cycle. This motivated us to introduce, in this

paper, a proposal for a systematic verification methodology we call Model-Combinatorial based testing (MCbt) to be used in testing safety-critical systems and produce safety evidences that contribute positively in the process of safety certification. The MCbt framework is an integration of model-based testing, combinatorial testing and safety analysis to perform testing activities at different system levels and testing targets. We presented MCbt tasks using SPEM 2.0 meta models and we performed an evaluation on the work product output of these tasks with respect to safety evidence type production. Based on the taxonomy provided by Nair et al. in [14], the evaluation was conducted by linking work product outputs of each MCbt task to a safety evidence using the definition of each testing activity that could be conducted using MCbt. MCbt showed preliminary results regarding safety evidence production with about 52% safety evidence coverage. In order to experimentally validate and evaluate MCbt scalability in testing safety-critical systems, in a future work, we will apply MCbt to different case studies from different safety-critical domains and investigate MCbt's contribution to the certification compliance process. This could also help identify more safety evidences that were not clear if they could be produced at this current level of analysis and maturity of MCbt.

ACKNOWLEDGMENT

This work was partially supported in part by NSF grant #1439693 to the University of Denver. The author B. Gallina is financially supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474).

REFERENCES

[1] J. Frost, "An Ada95 solution for certification of embedded safety critical applications," In: Gonzlez Harbour M., de la Puente J.A. (eds) Reliable Software Technologies Ada-Europe 99. Ada-Europe. Lecture Notes in Computer Science, vol. 1622, Springer, Berlin, Heidelberg 1999.

[2] A. Gannous, A. Anneliese and B. Gallina, "Bridging the gap between testing and safety certification," In IEEE Aerospace Conference, 2018.

[3] D. Cofer and S. Miller, "DO-333 Certification case studies," In: Badger J.M., Rozier K.Y. (eds) NASA Formal Methods, Lecture Notes in Computer Science, vol. 8430, Springer, Cham 2014.

[4] RTCA DO-178C, "Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc," Washington DC, 2013.

[5] L. Rierson, Developing safety-critical software: a practical guide for aviation software and DO-178C compliance. CRC Press, 2013.

[6] ISO 10218, "Robots and robotic devices – Safety requirements for industrial robots. International Standard," 2011.

[7] ISO 13482, "Robots and robotic devices – Safety requirements for personal care robots. International Standard," 2014.

[8] EN 50126, Railways Applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety," 2017.

[9] ISO 26262, "Road vehicles – Functional safety. International Standard," 2011.

[10] IEC 62304, "Medical device software software life cycle processes. International Electro-technical Commission," Geneva, 2006.

[11] R. Bloomfield and P. Bishop, "Safety and assurance cases: past, present and possible future an Adelard perspective," Making Systems Safer, Springer, London, pp 51-67, 2010.

[12] B. Monate, E. Ledinot, H. Delseny, V. Wiels and Y. Moy, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," IEEE Software vol. 30, p. 50-57, 2013.

[13] U. Zdun, A. Bener and E. L. Olalia-Carin. Introduction: Software Engineering for Compliance. In IEEE Software, vol. 29, no. 3, pp. 24-27, 2012.

[14] S. Nair, J. de la Vara, M. Sabetzadeh, and L. Briand, "An extended systematic literature review on provision of evidence for safety certification," Information and Software Technology, Vol. 56, Issue 7, pp. 689-717, 2014.

[15] C. Ericson, Hazard Analysis Techniques for System Safety. John Wiley & sons Inc., 2005.

[16] M. Sánchez and M. Felder, "A systematic approach to generate test cases based on faults," In Argentine Symposium in Software Engineering, Buenos Aires, Argentina, 2003.

[17] R. Nazier and T. Bauer, "Automated risk-based testing by integrating safety analysis information into system behavior models," In IEEE 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW), pp. 213-218, 2012.

[18] A. Gario, A. Andrews and S. Hagerman, "Testing of safety-critical systems: An aerospace launch application," 2014 IEEE Aerospace Conference, pp. 1-17, 2014.

[19] A. Anneliese, S. ELakeili, A. Gario, and S. Hagerman, "Testing proper mitigation in safety-critical systems: An aerospace launch application," In IEEE Aerospace Conference, 2015.

[20] M .Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," Software Testing, Verification & Reliability Journal, vol. 22, no. 5, pp. 297-312, 2012.

[21] A. Dias Neto, R. Subramanyan ,M. Vieira, and G. Travassos, "A survey on model-based testing approaches: a systematic review," 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies, pp. 31-36, 2007.

[22] M. Fantinato and M. Jino, "Applying extended finite state machines In software testing of interactive systems," International Workshop on Design, Specification, and Verification of Interactive Systems, Springer, 2003.

[23] C. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and combinatorial testing for effective test case generation," In Proceedings of the 2012 International Symposium on Software Testing and Analysis (ACM), pp. 100-110, 2012.

[24] M. Grindal, J. Offutt, and S. Andler, "Combination testing strategies: a survey," Software Testing, Verification and Reliability Journal, vol. 15, no. 3, pp. 167-199, 2005.

[25] Y. Lei and K. Tai, "In-Parameter-Order: A test generation strategy for pair-wise testing", Proceedings of the Third IEEE High Assurance Systems Engineering Symposium, IEEE Computer Society Press, pp. 254261, 1998.

[26] OMG, 2008, Software & systems process Engineering Meta-model (SPEM), v 2.0. Full Specification formal/08-04-01, Object Management Group.

[27] P. Ammann and J. Offutt, Introduction To Software Testing, 1st ed., Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013, USA, 2008.

[28] C. Ericson, "Concise Encyclopedia of System Safety: Definition of Terms and Concepts," Wiley, Hoboken, 2011.

[29] A. Abran, and J.W. Moore, "Guide to the software engineering body of knowledge," IEEE Comput. Soc, 2004.

[30] V. Hilderman, and T. Baghi, "Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware)," Avionics Communications, 2007.

[31] C. Cârlan ,B. Gallina , S. Kacianka and R. Breu, "Arguing on software-level verification techniques appropriateness," In: International conference on Computer Safety, Reliability, and Security. SAFECOMP 2017. Lecture Notes in Computer Science, vol 10488, pp. 39-54, Springer, Cham, 2017.

[32] B. Gallina, and A. Andrews, "Deriving verification-related means of compliance for a model-based testing process," Digital Avionics Systems Conference (DASC), IEEE/AIAA 35th, 2016.