Variability in Test Systems: Review and Challenges

Aitor Arrieta, Goiuria Sagardui, Leire Etxeberria

Computer and Electronics Department
Mondragon Goi Eskola Politeknikoa
Goiru 2, Arrasate-Mondragon, Spain
Email: {aarrieta, gsagardui, letxeberria}@mondragon.edu

Abstract—**Customizable products are increasing in our society, which creates a need on software systems, embedded systems and cyber-physical systems to handle variability. In addition, many companies are moving towards continuous integration and deployment and as a result, an automated solution to testing the relevant configurations is needed. Thus, variability appears in different stages of the product life-cycle. When validating these configurable systems, the test framework has to deal with their variability in order to test different system variants. Modelling variability in the test systems is an elegant solution to test and validate variability-intensive systems. This paper studies some test systems that handle variability and compares their characteristics and limitations, which can help test engineers needing a variability handling test system to choose among different approaches.**

Keywords–Test Systems; Test Architecture; Variability; Validation

I. Introduction

Variability is the ability to change or customize a system [1], and it can be understood as configurability or modifiability [2]. In the case of configurability, the variability appears in the product space, whereas in the case of modifiability, variability appears in the time space. The discipline of representing variability in models that describe the common and variable characteristics of a product is named variability modelling [3].

The different demands of the society and the users are some of the causes for customized products. As a consequence, variability in different points of the products, systems and development phases is increasing. Variability-intensive systems can be configured into thousands or millions of system variants. From one system variant to another, variability can appear not only in the product itself, but also in the elements in charge of testing the product, i.e., test system.

When testing different system variants, the test system has to be configured and adapted as well in order to test them. Due to this issue, modelling variability in the test system can be an interesting approach for testing variability-intensive systems, such as software product lines (SPLs), configurable embedded systems or configurable Cyber-Physical Systems (CPSs).

Most of the reviews and surveys in the field of variability modelling focus on the variability of the system itself, e.g., [4], or the use of variability modelling in industrial practice, e.g., [3], where the different notations used for modelling variability are analysed. This paper presents a review of variability-handling test systems. To carry out this study, we have systematically reviewed the documented approaches.

The rest of the paper is structured as follows: A brief introduction about the background of test systems is presented in Section II. Section III explains the methodology that has been used to systematically review the documented approaches in journals and conference papers. Section IV presents the obtained results after applying the search methodology; the documented approaches are explained and a discussion and analysis is provided. Section V defines a set of open challenges about variability modelling in test systems. Finally, Section VI summarizes the obtained conclusions.

II. Test Systems

A test system is a set of components that interact with the objective of testing the System Under Test (SUT). The complexity of a test system can vary depending on the overall test objectives and type of testing. Some of the tests are performed in simulation, e.g., Model-in-the-Loop (MiL) or Software-in-the-Loop (SiL) simulation, whereas other tests are performed in emulation, where additional hardware is needed, e.g., Hardware-in-the-Loop (HiL). Other test systems support test automation, where a test scheduler that decides which test case to execute is mandatory.

The organization of the group of components comprising the test system is called the test architecture [5], which specifies the interaction among the different elements of the test system and the SUT. A test architecture is a necessary artefact in test and validation activities so that verification and validation activities can be systematic, and it allows the reuse of test cases along the different test phases [5].

Test cases are part of the test system and provide information about the test execution. In Model-Based Testing (MBT), test cases are automatically generated either from the System Model, i.e., from the model of the SUT or from a test model [6]. When the test cases are executed, the test results have to be determined. This is typically performed by other elements of the test system, such as test oracles, which are mechanisms that analyse the SUT output and are able to decide the test result [6].

Modelling variability in the elements of the test systems allow the execution of tests under different conditions. Moreover, requirements of a configurable system varies from a system variant to another, and the test system has to be adapted in order to test different system variants, thus, variability in the components of the test system can help to achieve this goal.

III. Search Method

This paper collects the results of a systematically developed state of the art study about variability-handling test systems. To carry out this study systematically, the guideline presented in [7] has been taken as a base, which follows the presented steps below:

- Definition of the research questions
- Search process
- Inclusion and exclusion criteria
- Data collection
- Data analysis

A. *Definition of Research Questions*

As mentioned above, the scope of this study is to analyse the current state of the art in the field of variability handling test systems. The Research Questions (RQs) to carry out the goal were the following:

- **RQ1**: Which are the approaches documented that take into account variability in test systems?
- **RQ2**: Which kind of systems are tested with the selected approaches?
- **RQ3**: Which are the used modelling or programming languages?
- **RQ4**: Which are the used test strategies?
- **RQ5**: Which is the variability modelling approach?

B. *Search Process*

The search process has been a manual search by using search strings on different scientific databases with the aim of identifying conference proceedings and journal papers since 2008. The used databases have been IEEE Xplore, ACM digital library, Science Direct and Springer. Other places and sources such as proceeding of VALID 2014 or relevant PhD theses available on the Internet have been also used. Once identified some conference papers, journal papers and PhD theses on these databases, some references of the selected studies have also been used to detect new papers that are not available on the proposed databases.

With respect to search strings, the following keywords were used in order to find new papers:

- (“Variability” OR “Variant” OR “Modifiability” OR “Configurable”) AND (“Test System” OR “Validation Environments”)

C. *Inclusion and Exclusion Criteria*

The inclusion criteria for selecting a paper was the following:

- The publication should be “journal”, “conference proceeding”, or “PhD Thesis”.
- The reader should clearly deduce that the test system or architecture handles variability.

The exclusion criteria for excluding a paper was the following:

- The publication is not written in English.

D. *Data Collection*

The data extracted from each selected study was:

- Full reference (authors, title, journal or conference and year)
- Institution or institutions of the authors
- Characteristics and limitations of the proposed approach

E. *Data Analysis*

The data analysed once extracted the needed information was the following:

- Which is the main characteristic of the approach
- Which kind of systems are tested with the proposed approach
- Modelling tools used for the development of the test system
- Which are the main limitations of the selected approaches
- Variability points in the test system
- Variability modelling approach

F. *Search Result*

After applying the search strings in the aforementioned scientific databases, it concluded with 87 publications in IEEE, 12 in ACM, 1 in VALID 2014 and 1 known PhD thesis. However, most of the publications did not offer the expected overview. After following the inclusion and exclusion criteria, nine publications have been selected for the review, which are explained below. The selected publications are the following (addressing RQ1): [5][8][9][10][11][12][13][14].

IV. Variability in Test Systems

Testing variability-intensive systems is a very time and resource consuming activity due to the high number of possible variants. When a specific system variant has to be tested, the test system in charge of testing it has to be configured. The variability among the different system variants affects the test system, where different test cases have to be executed to validate a specific system configuration, and as a result, variability in the test model is required [15].

Variability handling systems appear in a wide range of systems, such as SPLs, embedded software and systems, mobile applications, CPSs, etc. The variability-handling test system approaches presented in the selected publications are used to test the following kind of systems (addressing RQ2):

- Non configurable Embedded Systems: [8][9]
- Variability handling systems (e.g., SPLs, highly configurable CPSs, etc.): [11][12][13][14][5]

TABLE I. COMPARATIVE TABLE OF THE SELECTED TEST SYSTEMS

Ref.	Modelling Language	SUT	Test Strategy	Variability Points	Variability Management	Variability Modelling
[8]	Messina	Embedded Systems	Evolutionary	Evolutionary algorithm configuration variables, target configuration	Not considered	Not used
[9]	Simulink	Embedded and CPSs	Functional Testing	In test stimuli generation	Not considered	Uniform variability
[5]	Simulink	Configurable CPSs	Functional Testing	SUT, Test oracles and Test data generator	Feature Models	Variant of negative variability
[11]	UML and UTP	SPLs	Not specified	SUT, Test context, Test cases, test component, data pool, data partition and data selector	UML sequence diagrams and a proposed UTP extension	UTP extension
[12]	UML and UTP	SPLs	Not specified	SUT, SUT interface, test context, data pool, data partition	OVM, UML and UTP	UTP extension
[13]	State Machines	SPLs	Regression testing	SUT, Test model, test goals, test suite and test plan	Not considered	Delta modelling
[14]	State Machines	SPLs	Incremental testing	SUT, Requirements, test model, test goals, test suite and test plan	Not considered	Delta modelling
[10]	Home-grown	General purpose software	Not specified	User interface, test control, code generator, information system and gateway	Not considered	Not used

- General Purpose Software: [10]

In addition, there are different tools, modelling or programming languages for developing these kind of systems. As a consequence, the modelling languages used for developing variability-handling test systems differ (addressing RQ3):

- MATLAB/Simulink: [9][5]
- State machines combined with delta modelling: [13] [14]
- UML-UTP: [11][12]
- Messina: [8]
- Home-grown: [10]

Other characteristics of test systems are the test strategies used to test the SUT. In the selected approaches, we have detected different test strategies (addressing RQ4): In [8] evolutionary testing is used, in [9] and [5] functional testing, in [13] regression testing and in [14] incremental testing. Other approaches do not consider the test strategy or it is not clear for the reader, e.g., [10].

With regard to variability modelling (addressing RQ5), different approaches are considered: Zander-Nowicka uses uniform variability modelling in [9]. Lity et al. [13] and Dukaczewski et al. [14] use delta modelling. Perez et al. use UML and UTP extension to model variability in [11] and [12]. In our previous work, we generate a skeleton model that acts as a core model, and when configuring a specific variant, the selected components of the skeleton model are replaced by models stored in a Simulink library, and the non selected are removed [5]. Other approaches ([10][8]) do not model variability. Table I summarizes the main characteristics of each approach.

A. Variability Handling Test Systems

The approach presented in [8] shows an evolutionary test system, primarily based on the MESSINA tool, that tests functional and non-functional properties of embedded systems. An evolutionary algorithm is an optimization technique based on the principles of the Darwinian theory of evolution, where a set of candidate solutions called individuals are selected. The fitness of these individuals are evaluated by the evolutionary algorithm by executing a problem-specific fitness function. The proposed approach by Kruse et al. in [8] supports MiL, SiL, Processor-in-the-Loop (PiL) or Hardware-in-the-Loop (HiL) test platforms, and allows the reuse of test cases across them. In the case of MiL and SiL test system configurations, MESSINA supports different tools, e.g., MATLAB/Simulink, ASCET models, etc. In the case of HiL, MESSINA is connected to modularHiL, a universal HiL test system developed by Berner & Mattner. The main variability points of this approach can be found in the configuration variables for the evolutionary algorithm, (e.g., mutation rate, crossover rate, etc.) as well as the test system target configuration, i.e., MiL, SiL, PiL or HiL.

Model-in-the-Loop for Embedded Systems Test (MiLEST) is a toolbox for MATLAB/ Simulink developed by Zander-Nowicka in [9]. This test system is designed towards the validation of automotive real-time embedded systems in Model-in-the-Loop (MiL). The hierarchy of MiLEST is divided into four abstraction levels: Test Harness level, Test Requirement level, Test Case level and Feature level. Although the approach in [9] proposes mechanisms for modelling variants, as shown in Figure 1, the test system itself is not designed for the validation of variability-handling systems. The proposed modelling technique is uniform variability. This variability modelling technique allocates all the components in the modelling framework, i.e., Simulink, and the variability is bound with different mechanisms, e.g., switch and constants. As there are unused components allocated in the simulation framework while simulation is running, simulation time is increased as explained in [16].

Our previous work [5] presents a configurable test architecture for the automatic validation of variability-intensive CPSs, together with a model-based process (Figure 2) for the systematic validation of these kind of systems. Variability of the test system is managed using the tool FeatureIDE [17] and saved into a *.xml file. This file is read by a test architecture generator that semi-automatically generates the skeleton of the test system. The tool FeatureIDE also allows generating different product configurations either automatically

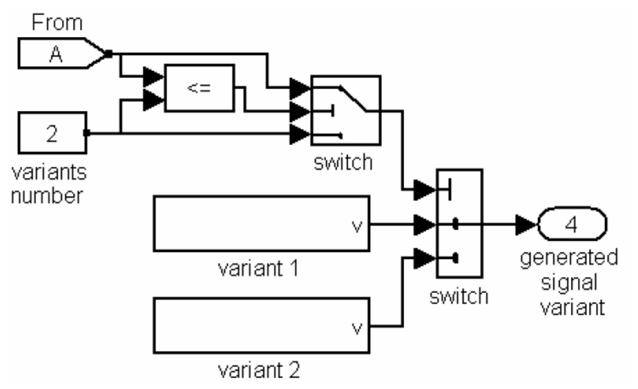

Figure 1. Variability Modelling Mechanism of MiLEST [9]

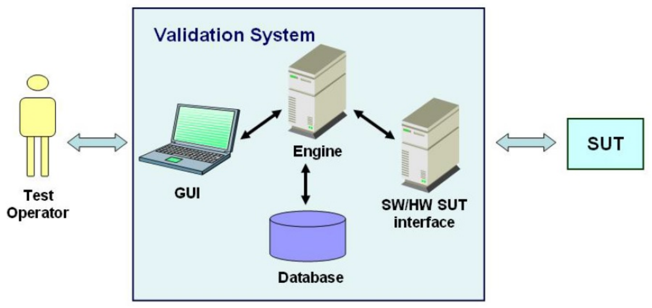

Figure 3. Test Elements Composing the Validation System [10]

(using pair-wise or t-wise techniques) or manually. These configurations are saved into a *.config file, which is read by the test configurator. The test configurator automatically configures the test system for the selected system configuration. Finally, the work describes the different variability points of the components of the test architecture: variability of the SUT, variability of the test data generator, variability of the test oracles and test control.

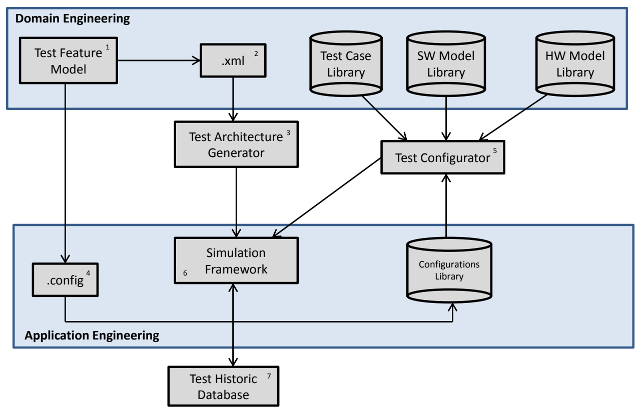

Figure 2. Model-Based Testing process for the systematic validation of variability-intensive CPSs [5]

A product line of validation environments with variability to test different applications in different domains and technologies is proposed in [10]. The study presents a validation environment able to test different SUTs from different domains, used programming languages, etc. Different elements of the validation system are identified (Figure 3) and the variability points together with variability requirements are identified and classified in a table.

The validation system proposed by [10] works as follows: The test engineer executes a test through the GUI, the GUI sends the test command to the engine, and this transforms the test command into the programming language that the SUT understands. For this step, the engine communicates with the database to obtain the correspondences between the source and target languages. When the transformation is finished, the command is sent to the SUT through the SUT interface, and awaits the response to begin the process again. These steps are shown in a UML sequence diagram depicted in Figure 4. The variability points of this system includes the user interface, test control, code generator, information system or gateway.

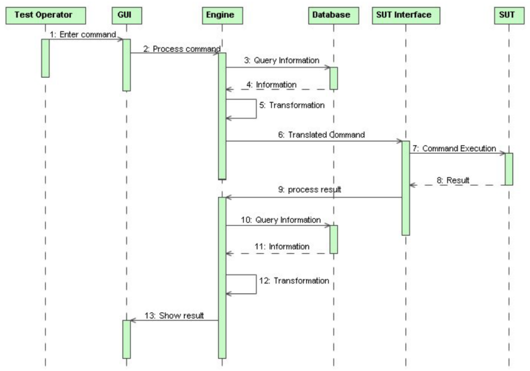

Figure 4. Interactions between the elements of the validation system proposed in [10]

The study presented in [11] defines an extended architecture for UTP to deal with variability in the test models, where the meta-model is shown in Figure 5. The proposed extension includes mechanisms to describe the behaviour of test cases and other elements needed to support variability. An example is illustrated in Figure 6.

The main variability in the proposed UTP extension is included in the Test Context, Test Cases, Test Components, UTP and Data Pool, Data Partition and Data Selector:

- TestContext: It is a class that organizes the test artifacts and contains test cases [11]. It can be stereotyped with "Variation Point", which means that the test cases corresponding to the TestContext have variation points [11].
- TestCase: A test case is represented with UML sequence diagram in [11]. A test case can also be stereotyped as "Variation Point" for testing a functionality with variability [11].

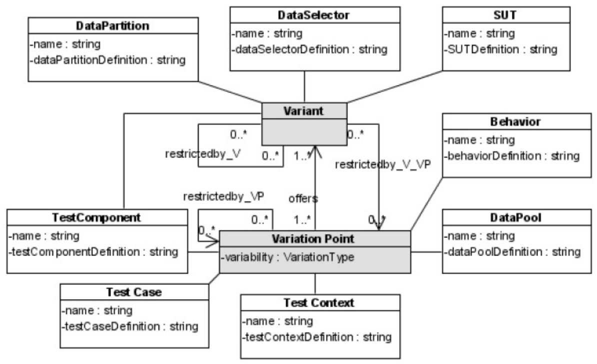

Figure 5. Proposed Extension to the UTP meta-model for handling variability [11]

- TestComponent: Test components interact with the SUT with the aim of realizing the test behaviour [11]. In the proposed extension, a test component can be stereotyped with “Variation Point” or “Variant”, which means that the test component can encapsulate the communication with the SUT, for the entire variation point or just for one of its variants.
- SUT: It can be stereotyped as “Variant”, which means that it realizes the functionality for its variant.
- DataPool, DataPartition and DataSelector: The DataPool contains the test data while the DataPartition the equivalence classes and data sets [11]. The dataPool can be stereotyped as “Variation Point”, which means that contains specific data for a Variation point. The DataPartition and the DataSelector are stereotyped as “Variant”, which means that the DataPartition contains the data associated with one of its variants and the dataSelector selects the data in the DataPartition for a specific variant.

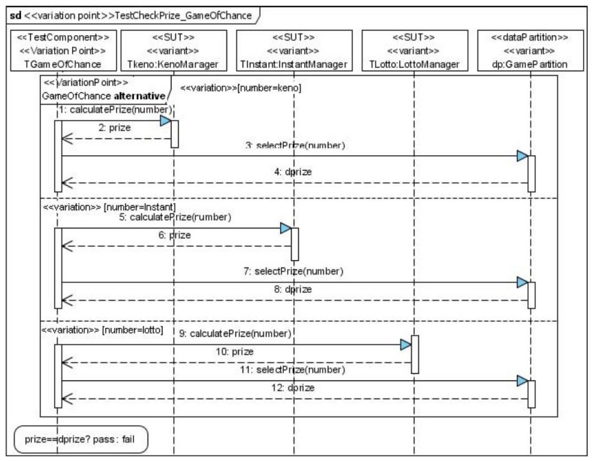
sd <<variation point>>TestCheckPrize_GameOfChance
<<TestComponent>> <<Variation Point>> TGameOfChance
<<SUT>> <<variant>> Tkeno:KenoManager
<<SUT>> <<variant>> TInstant:InstantManager
<<SUT>> <<variant>> TLotto:LottoManager
<<dataPartition>> <<variant>> dp:GamePartition
<<VariationPoint>> GameOfChance alternative
<<variation>>[number=keno]
1: calculatePrize(number)
2: prize
3: selectPrize(number)
4: dprize
<<variation>> [number=Instant]
5: calculatePrize(number)
6: prize
7: selectPrize(number)
8: dprize
<<variation>> [number=lotto]
9: calculatePrize(number)
10: prize
11: selectPrize(number)
12: dprize
prize==dprize? pass : fail

Figure 6. Example of a Test Case Using the UTP Extention proposed in [11]

In [12], a model-based method for the automatic generation of test cases for the testing of SPLs is described using UML 2.0, the UML Testing Profile and the QVT language. The approach differentiates two main models: Platform Independent Models (PIM) and Platform Independent Test Models (PIT). Each of the models are separated for the domain engineering layer (PIMD and PITD) or the application engineering (PIM and PIT), as shown in Figure 7. Variability of the system models are managed with an extension of the UML Testing Profile. The proposed approach uses Orthogonal Variability Model (OVM) for managing variability of the test system. In this case, variability in the test system can be found in the test case behaviour and in the test architecture. The test case behaviour is modelled using sequence diagrams handling variability, whereas the variability-handling test architecture is modelled with UML class diagrams. The test model is automatically transformed taking as source models the design model and the variability model using QVT.

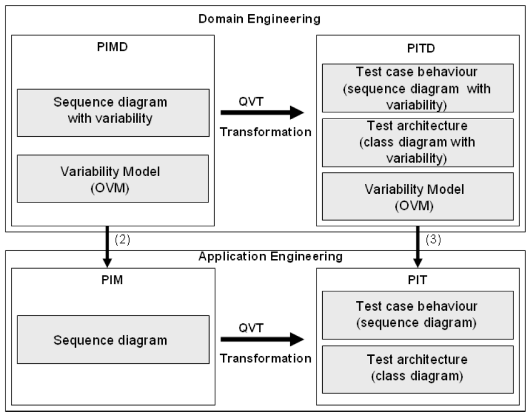
Domain Engineering
PIMD
Sequence diagram with variability
Variability Model (OVM)
QVT Transformation
PITD
Test case behaviour (sequence diagram with variability)
Test architecture (class diagram with variability)
Variability Model (OVM)
(2)
(3)
Application Engineering
PIM
Sequence diagram
QVT Transformation
PIT
Test case behaviour (sequence diagram)
Test architecture (class diagram)

Figure 7. Model driven testing approach for SPLs [12]

A Model-Based SPL regression testing approach is proposed in [13], where delta-oriented state machine as variable test models are used to incrementally evolve test artifacts by re-using artifacts of previously tested variants. Variability is applied in test artifacts, which are composed of (1) Test Models, (2) Test Goals, (3) Test Suite and (4) Test Plan. In [13], products evolve by applying deltas. Following the idea of Delta Modelling [18], a core model is developed and product variants are represented by the core model and a set of deltas that describe changes to be applied to this core model. From the testing point of view, a test artifact is developed to test the core system and deltas are applied to the test artifact in order to test the rest of the products.

Delta-oriented testing is also used in [14]. In this approach, Dukaczewski et al. propose a delta-oriented incremental testing approach based on textual requirements, where the test cases are directly associated to the requirements. Based on delta modelling [18], a delta describes how the behaviour of two system variants differs from each other. The main idea of this approach is based on testing the core system exhaustively and consider only the newly added or modified behaviour of the previous sytem during testing when moving to the next system variant [14]. Three steps are carried out to apply the proposed delta-oriented testing approach: The first step consist in selecting a core system, the requirements related to the selected system are separated from the requirements of all possible system variants and the selected core system is tested exhaustively. In the second a variant system is selected, deltas are applied to requirements to define changes between the requirements of different system variants by adding or removing requirements. Lastly, the test cases associated with the requirements are classified. Depending on the delta, the test cases are divided into four categories (Figure 8) [14]:

- Invalid: Test cases that become invalid for the new system variants. Invalid test cases belong to removed requirements.
- New: Test cases that are added to the new system variant, which belong to added requirements.
- Reuse: Test cases from the previous system variant that are not affected by the performed changes, which can be obtained using model slicing techniques (e.g. [19]). These test cases belong to unchanged requirements.
- Retest: Test cases from the previous system variant that are affected by the changes and have to be executed again. The test cases for retest are determined by the following identified options [14]: Random, Meta data, History/Statistics, Test expert and model.

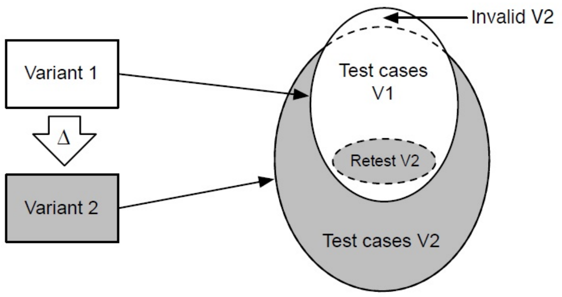

Figure 8. Delta Test-Sets [14]

B. Discussion and Analysis

The previous section has introduced the different documented approaches for handling variability in test systems. Each of these systems have their advantages and their limitations. When modelling variability in test systems, several characteristics are important, such as variability management, test automation, variability in test cases, or the specification of a test architecture. This section analyses the main limitations of each approach.

Kruse et al. propose a configurable test system for evolutionary testing of embedded systems in [8]. Variability in this test system appears in some configuration variables used by the evolutionary algorithm as well as the target configuration of the test system, i.e., MiL, SiL, PiL or HiL. Although the test system handles some variability points, it is not oriented for the validation of variability-handling systems.

Simulink is also used to model variability of test systems in two of the selected approaches ([9] [5]). This tool could be one of the most interesting when testing embedded software and CPSs, as it allows simulating the physical layer, as well as the cyber-digital layers (embedded system, software, etc.).

MiLEST is a toolbox oriented for the validation of embedded systems designed in [9], which can also be applied to CPSs. Although this test system shows variability modelling mechanisms, MiLEST is not designed for it. The variability points in the test architecture are limited to the test stimuli generator. Neither variability management tools nor automatic generation and configuration of the architecture for variability-intensive systems are used in this case. Another important factor when testing configurable systems is the simulation time. In this case, uniform variability is used as a variability modelling technique, which enlarges simulation time [16].

In our previous work [5], we analysed the variability of the test system and its components for the efficient validation of highly configurable CPSs. The test system is modelled in Simulink, and it is semi-automatically generated taking the information of a Feature Model into account. Apart from the SUT, variability can be found in the signals of the test data generator, requirements, test cases, signals of the test oracle, validation functions and validation function characteristics. In addition, we propose a traceability strategy among the features of the SUT and the test system. This strategy enables the automatic configuration of the test system depending on the selected SUT variant.

Apart from MATLAB/Simulink, other modelling languages are widely used when modelling embedded software, e.g., UML. In the case of [11], UML together with its UTP extension is used as a modelling tool. This approach analyses variability in several points, e.g., test context, test cases, etc. Moreover, some interesting concepts are provided that could be used in other test system, especially when modelling variability in test cases. In this case, variability is managed using UML models.

This UTP extension is used by the same author in [12]. In this case, the test models are generated automatically from the model of the SPL using the QVT Language. The variability modelling strategy of the test systems presented in [11] and [12] are clearly identified. In both cases, the test architecture is presented so that the interaction among the components of the test systems and the SUT is provided.

Delta modelling is used to model variability in [13] and [14]. In [13], Lity et al. propose a regression testing approach to test SPLs. The variability points of the test system can be found in test models, test goals, test suite and test plan. With regard to the drawbacks of this approach, on the one hand, the variability management of the test system is not specified. On the other hand, a test architecture is not considered, and as a consequence, the interaction among the components of the test system and the SUT cannot be appreciated.

Dukaczewski et al. propose incremental testing for the validation of SPLs in [14]. The way the test cases are classified in [14], “invalid”, “new”, “reuse” and “retest”, are an interesting option when testing SPLs. However, the proposed approach shows the same drawback as in [13], i.e., a variability management tool to model variability of the test system is not specified, and a test architecture showing the interactions among the test system and test components is not provided.

In the case of [10], a product line of validation environments is proposed. Although the interaction among the elements of the validation systems seems interesting, the analysed variability points of the systems are related to high-level elements, i.e., it does not take into account variability in test cases, test oracles, etc. Moreover, important characteristics such as variability management or test automation are not provided in this approach.

V. Open Challenges

Variability is an issue that has to be considered across the software-rich systems life cycle. In the previous section we have selected eight approaches that consider variability in the test systems. Nevertheless, there are still a lot of open questions and challenges when developing variant-rich test systems.

One of the major challenges would be to integrate a test system that could be able to test variant-rich systems from different domains. However, this is a complex issue, as different domains might need different kind of test systems. Furthermore, this can be infeasible because of the use of Domain Specific Languages, which warrants the use of different tools, e.g., SCADE for railway domain or MATLAB/Simulink for the automotive domain. Not only that, many variability points would have to be considered, as variability can appear in several points depending on the system.

Some of the selected test systems proposed a testing strategy. Depending on the validation phase, one test strategy could be more appropriate than another. Considering variability in the test strategy could be an interesting option, so that one test strategy or another could be chosen depending on the test needs and the validation stage.

A unified methodology that would warrant a systematic validation process of variant-rich systems of different domains could help test engineers with the validation activities. For instance, our previous work [20] proposes a model-based testing methodology for the validation of highly configurable CPSs.

One of the major problems in the validation of variant-rich systems is that as it is infeasible to test all the possible product configurations, the notion of the achieved test coverage is unclear. As a result, the analysis of new test metrics is a clear challenge in this field.

VI. Conclusion

This paper presents the current trends when modelling variability in test systems, issues that have to be considered as well as future challenges. Most of the research in the field of variability modelling of SPLs and variant-rich software focuses on the system itself. With regard to testing and validating SPLs and variant-rich software, most of the papers of the current state of the art propose generation of efficient configurations of the systems using different techniques such as combinatorial interaction testing (CIT). Other research efforts in this field consider the efficient test case generation for specific product variants. Although the research efforts in the field of variability modelling of test systems is not major, it is an important field of validation of variant-rich systems.

The paper has presented different approaches for testing different types of targets. In particular, the approaches presented in [9][8][5] are oriented to the testing and validation of real time embedded systems, embedded software or CPSs. On the other hand, the approaches presented in [11][12][13][14] have as testing objectives SPLs.

Some of the selected works do not consider variability management, e.g., [9][10]. This is not a problem if the test system is not variant rich, but in the case there are many variability points, the lack of a variability management tool can become a problem. In addition, the variability management tool can help to trace the variability of the test system with the variability of the SUT, as proposed in [5].

The paper also shows that variability in test systems is not just used to test variability-handling systems, but also general purpose systems. Three of the selected approaches consider variability in their test system although the SUT does not present any variability point. In the case of [8] and [9], the test systems are designed to test embedded systems, whereas the approach presented in [10] tests general purpose software.

VII. Acknowledgements

This work has been developed by the embedded systems group from Mondragon Goi Eskola Politeknikoa, supported by the Department of Education, Universities and Research of the Basque Government.

References

[1] J. V. Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in Proceedings of the Working IEEE/IFIP Conference on Software Architecture, ser. WICSA '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 45–54.

[2] S. Thiel and A. Hein, "Modelling and using product line variability in automotive systems," IEEE Software, vol. 19, no. 4, 2002, pp. 66 – 72.

[3] T. Berger, et al., "A survey of variability modeling in industrial practice," in Variability Modelling of Software-intensive Systems (VaMoS), 2013, pp. 7:1–7:8.

[4] J. Weiland and P. Manhart, "A classification of modeling variability in simulink," in Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems, ser. VaMoS '14. New York, NY, USA: ACM, 2014, pp. 7:1–7:8.

[5] A. Arrieta, G. Sagardui, and L. Etxeberria, "A configurable test architecture for the automatic validation of variability-intensive cyber-physical systems," in VALID 2014: The Sixth International Conference on Advances in System Testing and Validation Lifecycle, 2014, pp. 79–83.

[6] J. Zander-Nowicka, I. Schieferdecker, and P. J. Mosterman, A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains. Model-Based Testing for Embedded Systems, 2011, ch. 1, pp. 3–22.

[7] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," Inf. Softw. Technol., vol. 51, no. 1, Jan. 2009, pp. 7–15.

[8] P. M. Kruse, J. Wegener, and S. Wappler, "A highly configurable test system for evolutionary black-box testing of embedded systems," in Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ser. GECCO '09. New York, NY, USA: ACM, 2009, pp. 1545–1552.

[9] J. Zander-Nowicka, "Model-based testing of real-time embedded systems in the automotive domain," Ph.D. dissertation, Technical University Berlin, 2008.

[10] B. Magro, J. Garbajosa, and J. Perez, “A software product line definition for validation environments,” in 12th International Software Product Line Conference (SPLC), Piscataway, NJ, USA, 2008, pp. 45 – 54.

[11] B. Pérez, M. Polo, and M. Piattini, “Towards an automated testing framework to manage variability using the uml testing profile,” in AST, 2009, pp. 10–17.

[12] B. Pérez, M. Polo, and I. García, “Model-driven testing in software product lines,” in Proceedings of the 2009 IEEE International Conference on Software Maintenance (ICSM 2009), 2009, pp. 511 – 514.

[13] S. Lity, M. Lochau, I. Schaefer, and U. Goltz, “Delta-oriented model-based spl regression testing,” in 3rd International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2012, Piscataway, NJ, USA, 2012, pp. 53 – 6.

[14] M. Dukaczewski, I. Schaefer, R. Lachmann, and M. Lochau, “Requirements-based delta-oriented spl testing,” in 4th International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2013, San Francisco, CA, United states, 2013, pp. 49 – 52.

[15] D. Streitferdt et al., “Model-based testing of highly configurable embedded systems in the automation domain,” International Journal of Embedded and Real-Time Communication Systems, 2011, pp. 22–41.

[16] A. Arrieta, G. Sagardui, and L. Etxeberria, “A comparative on variability modelling and management approaches in simulink for embedded systems,” in V Jornadas de Computación Empotrada, ser. JCE 2014, no. 26-33, 2014.

[17] T. Thuem, C. Kastner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, “Featureide: An extensible framework for feature-oriented software development,” Science of Computer Programming, vol. 79, 2014, pp. 70 – 85.

[18] I. Schaefer, “Variability modelling for model-driven development of software product lines,” in VaMoS, 2010, pp. 85–92.

[19] J. Kamischke, M. Lochau, and H. Baller, “Conditioned model slicing of feature-annotated state machines,” in Proceedings of the 4th International Workshop on Feature-Oriented Software Development, ser. FOSD ’12. New York, NY, USA: ACM, 2012, pp. 9–16.

[20] A. Arrieta, G. Sagardui, and L. Etxeberria, “A model-based testing methodology for the systematic validation of highly configurable cyber-physical systems,” in VALID 2014: The Sixth International Conference on Advances in System Testing and Validation Lifecycle, 2014, pp. 66–72.