# Towards Incremental FSM-based Testing of Software Product Lines

**Maykon Luís Capellari[1], Itana Maria de Souza Gimenes[1], Adenilso da Silva Simão[2], Andre Takeshi Endo[2]**

[1]State University of Maringa (DIN-UEM)
Maringa – PR – Brazil

[2]Universidade de São Paulo (ICMC-USP)
São Carlos, SP, Brazil

{maykon, itana}@din.uem.br, {adenilso, aendo}@icmc.usp.br

***Abstract.*** *Software Product Line (SPL) is an approach which offers several benefits for organizations, such as significant reductions in the development and maintenance costs, reduced time-to-market, and personalized software products. In SPLs, the testing activity presents challenges due to characteristics of their development process. The cost of testing SPL is usually higher than the cost of testing traditional systems. SPLs foster the reuse of artifacts that include requirement specifications, code and models. Among different models used in an SPL, state-based models, such as Finite State Machines, are promising candidates to support the test case generation. Therefore, we propose a strategy to reuse test cases generated for different products of an SPL. Test cases are derived from Finite State Machines representing products instantiated from an SPL. The test cases generated for a product are reused when testing further products instantiated from the same SPL, in order to reduce the size of additional test cases. We illustrate our strategy in a case study using two SPLs of embedded system applications.*

## 1. Introduction

Client dissatisfaction and lack of software flexibility require software development processes that foster high productivity, offer new choices, differentiate products and reduce costs. In this scenario, an approach for software development, named *Software Product Line* (SPL), was proposed [Clements and Northrop, 2001; Gomaa, 2004]. An SPL is defined as a family of software systems that shares a common and manageable set of features that satisfies the specific needs of a particular area. The use of SPLs in companies produces several advantages, such as productivity improvements, product quality improvements, cost reduction, time-to-market reduction, and high software component reuse [SEI, 2011].

The members of an SPL are derived from a specific configuration of core assets. An SPL is based on two main concepts: mass customization and common platforms [Pohl et al., 2005]. The mass customization is the large-scale production of products adapted to individual client needs, while common platforms establish a resource base, from which other technologies and processes are built up. The essential activities of an

SPL approach are: core assets development (Domain Engineering), products development (Application Engineering), and the SPL management [SEI, 2011].

Testing strategies provides trustworthiness in a product by means of executing its code or part of it [Tevanlinna et al., 2004]. SPL testing techniques are similar to the ones applied to traditional software, such as unit testing, integration testing, and regression testing. However, the testing process is applied in two steps: domain engineering and application engineering, making the cost of testing SPL more expensive. Due to the complexity and the high number of possible product configurations of an SPL, the tests during the domain engineering are feasible only with the common and most used features. During the application engineering, system, integration and regression tests are performed to assure the quality of the final product according to the selected features.

As there are several UML-based methods for the representation of SPL artifacts [Gomaa, 2004], model-based strategies should be taken into account when defining testing techniques for SPLs. Model-based testing (MBT) is an approach in which existing models of the software are used to support the test case generation [Dalal et al., 1999]. State-based models, such as Finite State Machines (FSMs) [Gill, 1962], are mainly used to generate test cases that exercise sequences of events in a software, which also apply to SPLs. MBT and FSMs have been applied in various domains, such as network protocols and embedded systems [Broy et al., 2005; Zander et al., 2011]. Nowadays, most of the embedded systems are known to be reactive as they respond to events (inputs) and produce observable actions (outputs). State machines have been used to model these systems, such as the ones embedded in airplanes, cell phones, and domestic devices.

Several test case generation methods for FSMs have been proposed, such as W [Chow, 1978], HSI [Luo et al., 1994; Petrenko et al., 1994] and P [Simão and Petrenko, 2010]. The test sequences are applied to the implementation and the outputs are compared to the specification. The cited methods are known to generate complete test suites, i.e., all faults in a given fault domain are detected by those test suites. Although there are several approaches to test SPLs, a challenge in the testing process is the reuse of tests among instantiated products [Pohl and Metzger, 2006]. Moreover, keeping the test suite complete would also be desirable.

In this paper, we consider functional testing, which use the information available in models and specifications to decide which test cases are more relevant. For this, we present a testing strategy, called FSM-based Testing of Software Product Lines (FSM-TSPL), that focuses on the reuse of FSM-based test cases of instantiated products from an SPL. The FSM-TSPL strategy aims to test products instantiated from an SPL, by promoting the reuse of test suites between these products. To ensure compliance between the specification and the implementation of the current product under test, the P method is used to increment the test suites of previously-tested products into a complete test suite. The feasibility of the proposed strategy is shown based on a case study in the domain of embedded systems.

The paper is organized as follows. Section 2 presents definitions and notations of FSMs. Section 3 emphasizes the contribution of the proposed strategy as compared to related work. Section 4 describes the proposed strategy to test SPLs based on FSM-

based testing. Section 5 presents a case study. Finally, in Section 6, we present some conclusions and discuss future work.

## 2. Background

An FSM is a deterministic Mealy machine, which can be defined as follows.

**Definition 1**. An FSM $M$ is a 7-tuple $(S, s_0, I, O, D, \delta, \lambda)$, where $S$ is a finite set of states with the initial state $s_0$, $I$ is a finite set of inputs, $O$ is a finite set of outputs, $D \subseteq S \times I$ is a specification domain, $\delta : D \rightarrow S$ is a transition function, and $\lambda : D \rightarrow O$ is an output function.

Tuple $(s, x) \in D$ is a defined transition in state $s$ that consumes input symbol $x$. A sequence $\alpha = x_1 \ldots x_k$, $\alpha \in I^*$, is an input sequence defined for state $s \in S$, if there exist $s_1,\ldots,s_{k+1}$ such that $s = s_1$ and $\delta(s_i,x_i) = s_{i+1}$ for all $1 \leq i \leq k$. Notation $\Omega(s)$ is used to denote all input sequences defined for state $s$ and $\Omega_M$ as an abbreviation for $\Omega(s_0)$. Therefore, $\Omega_M$ represents all defined sequences for FSM $M$. The empty sequence is denoted by symbol $\varepsilon$. Notation $\alpha\beta$ represents the concatenation of the two sequences, $\alpha$ and $\beta$. Given a sequence $\alpha$ and a set of sequences $D$, $\alpha.D = \{\alpha\beta \mid \beta \in D\}$. A sequence $\alpha$ is prefix of a sequence $\beta$, $\alpha \leq \beta$, if $\beta = \alpha\omega$, for some sequence $\omega$. A sequence $\alpha$ is proper prefix of $\beta$, $\alpha < \beta$, if $\beta = \alpha\omega$ for some $\omega \neq \varepsilon$. Given a test set $T$, the notation $pref(T)$ represents all the prefixes of sequences in $T$, i.e., $pref(T) = \{\alpha \mid \beta \in T \text{ and } \alpha \leq \beta\}$. If $T = pref(T)$, then we say that $T$ is prefix-closed.

An FSM can be classified as:

- *Complete*: If an FSM has defined transitions for each input symbol to all states, i.e., $D = S \times I$. Otherwise, the FSM is partial;
- *Deterministic*: If, for each state, there exists at most one defined transition for each input symbol. Otherwise, the FSM is nondeterministic;
- *Strongly connected*: If every state is reachable from all the other states via one or more transitions. An FSM is initially connected if every state is reachable from the initial state $s_0$.
- *Distinguishable*: Two different states $s_i, s_j \in S$ are distinguishable if there exists a sequence $\gamma \in \Omega(s_i) \cap \Omega(s_j)$, given that $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$;
- *Minimal*: An FSM $M$ is reduced or minimal if all states are pairwise distinguishable.

It is assumed a reset operation that takes both the FSM and its implementation to their initial state. The reset must be inserted at the beginning of each test sequence, so the number of resets is equal to the number of sequences in a test suite. A test case of $M$ is a defined input sequence $\alpha \in \Omega_M$. A test suite of $M$ is a finite set of test cases of $M$, such that there are no two test cases $\alpha$ and $\beta$ so that $\alpha$ is prefix of $\beta$. Considering a specification $M_S$ with $n$ states and an implementation $M_I$ with $m$ states, $m \geq n$, a certain method has full fault coverage if it is capable of generating a test suite capable of detecting all the faults in any implementation [Simão et al., 2009]. This test suite is called $m$-complete and in the particular case $m = n$ is called $n$-complete. The length of a sequence $\alpha$ is represented by $|\alpha|$. This notation was extended to represent the length of

all sequences in a test suite $T$ , $|T|$. We denote by $\Im$ the set of all deterministic FSMs with the same input alphabet as $M$ for which all sequences in $\Omega_M$ are defined [Simão and Petrenko, 2010]. Set $\Im$ is a fault domain for $M$.

A set of input sequences $Q$ is a state cover of $M$ if for each state $s_i \in S$, there exists a sequence $\alpha_i \in Q$ that transfers the FSM from the initial state to $s_i$. This set includes the sequence ε to reach the initial state. A set of input sequences $P$ is a transition cover of $M$ if for each transition $(s, x) \in D$ there exist the sequences $\alpha, \alpha x \in P$ such that $\delta(s_0, \alpha) = s$. Set $P$ also includes the sequence ε [Fujiwara et al., 1991].

A characterization set, also known as $W$, is a set of defined input sequences that contains at least a sequence that distinguishes each pair of states in the FSM. In other words, for any two $s_i, s_j \in S, i \neq j$, there exists a sequence $\alpha \in W$ such that $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. A separating family is a set of state identifiers $H_i$ for a state $s_i \in S$ that satisfies the following condition [Luo et al., 1994]:

- for any two different states $s_i$ and $s_j$, there exist sequences $\beta \in H_i$ and $\gamma \in H_j$ that have a common prefix $\alpha$ such that $\alpha \in \Omega(s_i) \cap \Omega(s_j)$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$.

There exist several methods proposed in the literature that generate $n$-complete test suites, such as:

- The W method [Chow, 1978] uses the transition cover set $P$ to reach the states and transitions and the characterization set $W$ for state identification;
- The HSI method [Luo et al., 1994] uses the separating family to check the states as in the state identification as in the transition testing; and
- The P method [Simão and Petrenko, 2010] was initially proposed to increment a previously established test suite until reach an estimate $p$ $(p \leq n)$. However, if we consider that the initial test suite $TS_{init}$ is a set with the empty sequence $TS_{init} = \{\varepsilon\}$, a final $p$-complete test suite will be generated.

## 3. Related Work

Testing in SPL basically consists of reuusing testing techniques for single systems, such as unit testing, integration testing, and regression testing. However, these approaches consider that the testing process is divided into two phases: (i) tests in the domain engineering and (ii) tests in application engineering. In domain engineering, domain artifacts and common features are tested. In application engineering, other different tests are performed according to the selected features of the product. A complete survey on SPL testing can be found in [Oster et al., 2011].

Some important approaches to SPL testing are:

1. Olimpiew and Gomaa [2009] propose a method to create test specifications from use cases and feature models. The method aims to reduce the number of reusable test specifications, which are created to cover all use case scenarios, all features, and selected feature combinations of an SPL.

2. Bertolino and Gnesi [2003] propose the Product Line Use Case Test Optimization (PLUTO), a method used to manage the testing process of an SPL. The authors used Product Line Use Cases, which is an extension of Cockburn's Use Case (a notation based in description of requirements in natural language [Cockburn, 2000]). This method is based on category partition; thus, it can be used to derive a generic test specification to SPLs, and a set of test scenarios to the application for a specific client.
3. Uzuncaova et al. [2007] use specifications given as formulas in Alloy, a first-order logic based on relations [MIT, 2010]. Alloy formulas can be checked for satisfiability using the Alloy Analyzer. The analyzer translates an Alloy formula to a propositional formula and finds an instance using an off-the-shelf SAT solver. Each program in a product line is specified as a composition of features, where each feature represents an Alloy formula. Tests are generated by solving the resulting formula.
4. Uzuncaova et al. [2008] propose an approach based on specification to generate tests for products of an SPL. Given properties of features as first-order logic formulas, this approach uses SAT-based analysis to automatically generate test inputs for each product in an SPL. To ensure soundness of generation, it was introduced an automatic technique for mapping a formula that specifies a feature into a transformation that defines incremental refinement of test suites.
5. Im et al. [2008] propose an approach used to automating test case definition in the context of a model driven approach to the SPL development. Test cases are automatically extracted from use cases, which are specified using a domain specific language (DSL). The structure of the DSL and proven patterns of test design provide the clues necessary to automatically extract the test cases. A chain of model-driven tools is used to automate the system testing process, which begins with a use case model and ends with automatic execution of system tests.
6. Cabral et al. [2010] propose an approach, called FIG Basis Path method, that translates a feature model into a feature inclusion graph. The features are associated with a set of test cases and, then, a walk in this graph is performed to generate products that will cover the graph for testing.

In our strategy, we used FSMs to model the products instantiated of the SPL, test suites are then generated from these models. Our strategy also takes advantage of the test suites generated for other products, reusing and incrementing them, ensuring conformity of the products. The test suite can be generated using three methods: the user-defined test suite, the HSI method [Luo et al., 1994; Petrenko et al., 1994], and the P method [Simão and Petrenko, 2010], as described in the next section.

## 4. Incremental FSM-based Testing of Software Product Lines

The FSM-TSPL strategy is applied during the instantiation of SPL products. No assumption is made about the SPL development approach and the strategy can be applied in an SPL developed using any approach, such as FODA [Kang et al., 1990], PuLSE [Bayer et al., 1999], and PLUS [Gomaa, 2005].

A domain engineer provides the requirements to start the SPL. In this stage, analysis models and the SPL architecture are produced, which together with the requirements are stored in the SPL repository. During the Application Engineering, the application requirements are provided and the specific products of SPL are configured from the repository templates. If requirement's problems and/or errors occur, they will be corrected in the activity of Engineering SPL in which these problems happen, otherwise, it generates an executable application initiating the testing phase of application. In the testing phase using FSM-TSPL, the tester designs an FSM representing the product to be tested.

The basis for the incremental test strategy is the use of algorithms for generating test sequences, which are based on the FSM specifications. Thus, our approach uses the HSI method [Luo et al., 1994] and P method [Simão and Petrenko, 2010] to generate test cases.

Consider an SPL under test, from which a set of products can be instantiated. We refer to a product of the SPL as $prod_i$. In FSM-TSPL, we assume that, after the instantiation, a product is tested using an FSM. Notation $M_i$ represents the FSM model for $prod_i$. Notation $M_j$ represents the FSM model for the next product to be tested $prod_{i+1}$. We also assume that there exists an initial test suite $TS_i$ derived from $M_i$. The test suite $TS_j$ represent the tests for $M_j$. $TS_j$ can be generated using a test case generation method or by an ad-hoc approach.

The testing strategy consists of the following steps (Figure 1):

1. Create an FSM for each product: In this step the tester creates an FSM model $M_i$ for a product $prod_i$ instantiated from SPL. The model can be designed by the tester or derived from artifacts of SPL. The FSMs developed for products of an SPL are usually deterministic and partial, given the large number of inputs and transitions, from SPL variabilities.
2. Select the FSM model: After created the FSM models, the tester should selected one to test.
3. User-defined Test Suite: This step only occurs if the tester wants to create manually a test suite. Therefore, it will generate test suite from an empty test case.
4. Generate Test Suite with HSI method: HSI method is used to generate test cases for each product of SPL. This step only occurs if the tester does not want to setup a test suite.
5. Verify Test Suite: After having defined test suite $TS_i$, in this step the strategy verifies whether test cases generated from FSM $M_i$ are defined for $M_j$ .
6. Refining Test Suite: test cases which are not defined in FSM $M_j$ are removed, i.e., $TS_j = TS_i \cap \Omega_{Mj}$.

7. Generate Test Suite with P method: After the test cases are checked, $TS_j$ will be used in the new test case generation for the P method. Thus, the P method reuses the test cases generated previously and aims to generate a smaller test suite.

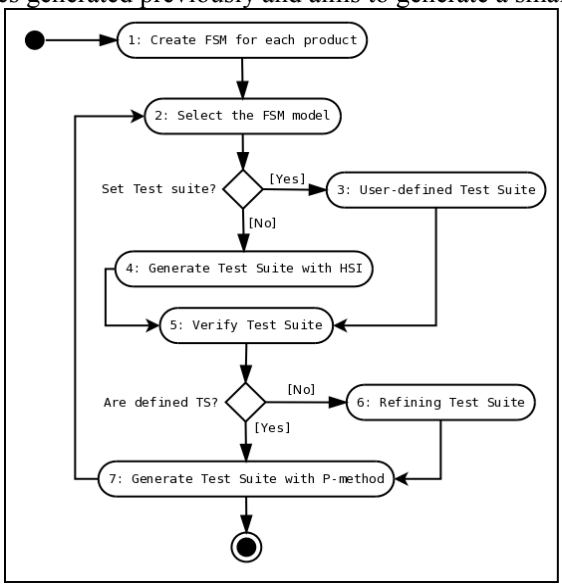


**Figure 1. Testing strategy for SPLs using FSM models.**

In order to facilitate the understanding of the FSM-TSPL strategy, Arcade Game Maker (AGM) [SEI, 2011] is used as an SPL example. AGM is a pedagogical SPL that produces some arcades games, such as Brickles, Pong, and Bowling. AGM was created by the SEI and contains a set of UML models and documents that facilitate the learning of SPL concepts. The games are controlled by one player, aiming to get more points by hitting obstacles on the screen. The feature model is depicted in Figure 2.

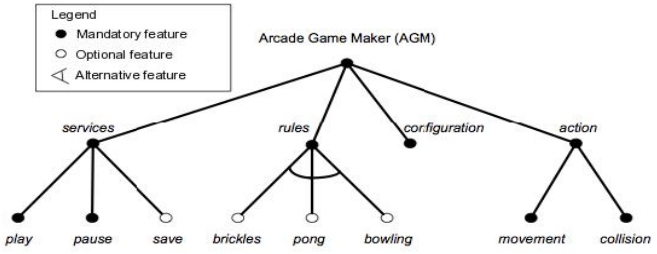


**Figure 2. Feature model for AGM.**

According to the Step 1, $M_1$ and $M_2$ are designed to specify $prod_1$ and $prod_2$, respectively, both instantiated from AGM. FSM $M_1$ has only common characteristics and represents the specification of game Brickles, this is illustrated in Figure 3.

FSM $M_2$, also specifies the Brickles game and contains the common features and an optional feature, save game. This optional feature adds a new state *SV* to FSM model e some transitions are changed as represented by Figure 4.

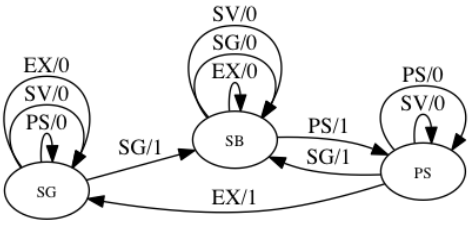


**Figure 3. FSM for product *prod*$_1$ of AGM.**

FSM $M_1$ was selected for the tests as Step 2. This FSM is the first product to be tested; thus, we can skip Step 3, since the test suite will be generated in the next step. The HSI method is applied on $M_1$ for the generation of test suite as described in Step 4.

The HSI method was applied on $M_1$, generating the following test suites: $Q$ = { ⟨ε⟩,⟨SG⟩,⟨SG,PS⟩ }, $P$ = { ⟨ε⟩, ⟨SG⟩, ⟨SG,PS⟩, ⟨PS⟩, ⟨SV⟩, ⟨EX⟩, ⟨SG,SG⟩, ⟨SG,SV⟩, ⟨SG,EX⟩,⟨SG,PS,SG⟩,⟨SG,PS,PS⟩,⟨SG,PS,SV⟩,⟨SG,PS,EX⟩ } and the sets $H_i$ are: $H_0$ ={⟨EX⟩,⟨PS⟩}, $H_1$ ={⟨EX⟩,⟨PS⟩} e $H_2$ = {⟨EX⟩}. From these sets, it is generated the test suite $TS_{HSI}$ with size 72.

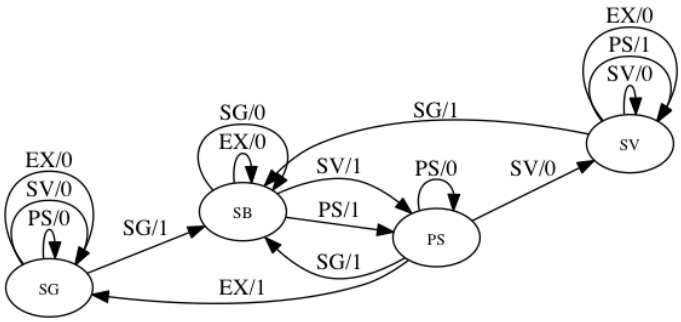


**Figure 4. FSM for product *prod*$_2$ of AGM.**

Applying the HSI method in the FSM $M_2$, we obtain the following test suites: $Q$ = { ⟨ε⟩,⟨SG⟩,⟨SG,SV⟩,⟨SG,SV,SV⟩ }, $P$ = {⟨ε⟩, ⟨SG⟩, ⟨SG,SV⟩, ⟨SG,SV,SV⟩, ⟨SV⟩, ⟨PS⟩, ⟨EX⟩, ⟨SG,PS⟩, ⟨SG,SG⟩, ⟨SG,EX⟩, ⟨SG,SV,SG⟩, ⟨SG,SV,PS⟩, ⟨SG,SV,EX⟩, ⟨SG,SV,SV,PS⟩,⟨SG,SV,SV, SV⟩, ⟨SG,SV,SV,SG⟩,⟨SG,SV,SV,EX⟩ } and the sets $H_i$ are: $H_0$ = {⟨EX⟩,⟨PS⟩,⟨SV⟩}, $H_1$ = {⟨EX⟩,⟨SV⟩}, $H_2$ = {⟨EX⟩,⟨PS⟩,⟨SV⟩} e $H_3$ = {⟨EX⟩}. From these sets it is generated the test suite $TS_{HSI}$ with size 143.

After the generation of the test sequences for FSM $M_1$, FSM $M_2$ is the selected in Step 2 for the next iteration, and use the test sequences generated from FSM $M_1$, thus it is verified if the test sequences are specified in FSM $M_2$ as Step 5. In Step 6, the test cases that are not defined in FSM $M_2$ are removed; in this case all remaining test sequences are defined by the FSM.

The generation of test suites of the P method, Step 7, which has as input a file containing the specification of the FSM. If there exists an initial test suite, the P method

increments the test suite until it covers the FSM, otherwise it generates the test suite from the empty test suite.

For the generation of the test suite out of FSM $M_2$, Step 7, the use of the generated test FSM $M_1$, so that the P method uses these tests and increases to the level of coverage satisfactory.

After the generation of test cases using HSI, we apply the P method in FSM $M_2$ and use the test cases of FSM $M_1$, generated by the HSI, as input to the test suite P method. Thus we obtain a test suite with size 104.

In the FSM-TSPL strategy, the tests were initiated with the set HSI and P, respectively. The test suites were generated as follows: To the first product it was considered test suites generated by the HSI and P methods, in this way cost was not considered; the second product was tested with test suites generated from the FSM for the first product, the third product from the second product, and so on.

The test suites generated by the FSM-TSPL strategy have been refined (removed duplicates, sequences defined in the input set, and prefixes), thus generating a relatively smaller test suite.

## 5. Case study

In order to evaluate the feasibility of the proposed strategy, a case study was conducted using two SPLs, AGM and Mobile Media. We evaluated the test suite length (cost), the number of resets and the cumulative cost to each method. The evaluation of the test suite length was performed individually, product by product, while the cumulative cost was calculated by summing of the test suite length for all products (such as $|T_1| + |T_2| + \ldots + |T_n|$).

Products were instantiated for each SPL and an FSM model was designed for each product. The FSM was modeled manually based on the behavior of each product instantiated from SPL, and these have the following properties: complete, deterministic, reduced and initially connected. For each product, the HSI, P methods and the FSM-TSPL strategy were applied, the latter was divided in FSM-TSPL (HSI) which uses the test suite generated by the HSI method and FSM-TSPL (P) that uses the suite test generated by the P method. Moreover, the first product was not evaluated in the FSM-TSPL(HSI) and FSM-TSPL(P) strategy because there was not a previous test suite to be used as input for our strategy.

The algorithms used in the evaluation of the experiments were run on an Intel(R) Core(TM) 2 Duo 2.4GHz, 4GB of RAM and Mac OS X 10.6.8.

### 5.1. Methodology

The method used in this case study can be described by the following steps:

1. Creation of the FSM model for each product of LP;
2. Select an FSM to be tested;
3. Apply the HSI method in selected FSM;
4. Apply the P method in selected FSM;

5. Apply the FSM-TSPL strategy using the test suite generated by the HSI method in the selected FSM;
6. Apply the FSM-TSPL strategy using the test suite generated by the P method in the selected FSM;
7. Refine test set generated by the FSM-TSPL using HSI sets;
8. Refine test set generated by the FSM-TSPL using P sets;
9. Comparison of the generated sets;
10. Analysis of results.

### 5.2. AGM

Six products containing distinct characteristics were instantiated, they are instantiated from the Base (mandatory features), and for each product, alternative and optional features were selected. For each product an FSM was modeled; the number of states varies between 3 and 4, with four inputs and two outputs.

Figure 5 shows the lengths (cost) for test suites generated from FSM by the HSI method, the P method (using an empty initial set), and the FSM-TSPL strategies.

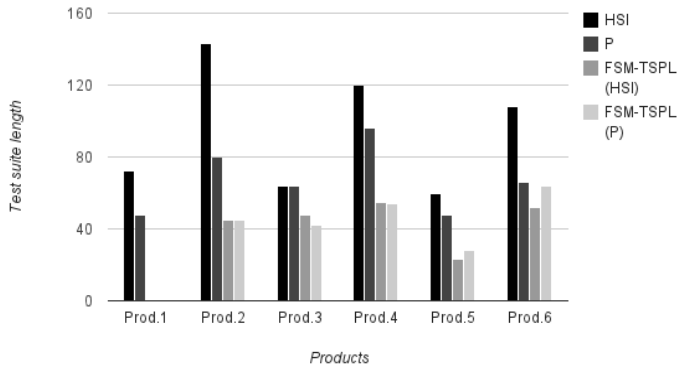


**Figure 5. Test suite length for AGM.**

Notice that the HSI method generated the largest sets compared with other methods, the P method sets showed relatively lower than the HSI and test suites generated from the FSM-TSPL strategy were lower compared with other methods.

The number of resets was also calculated based on the refined test suites. The relationship between the number of resets for each product may be seen in Figure 6. The HSI method has also generated more reset operations compared with other methods, the FSM-TSPL generated the lowest number of resets.

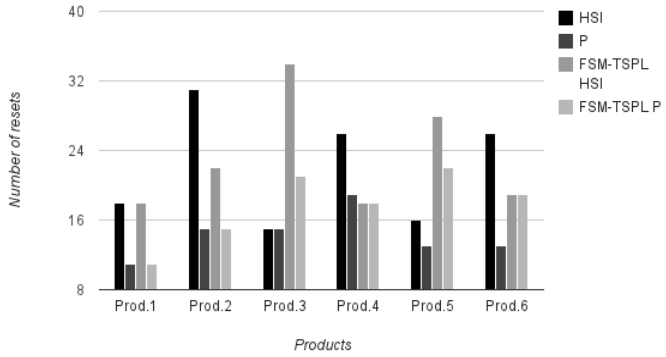


**Figure 6. Number of resets for AGM.**

Figure 7 presents the cumulative cost regarding the generation of test suites for each product. Observe that the HSI method has the highest cost to test all products, followed by P and FSM-TSPL (P). FSM-TSPL (HSI) has the lowest cost which is 223 inputs at the end.

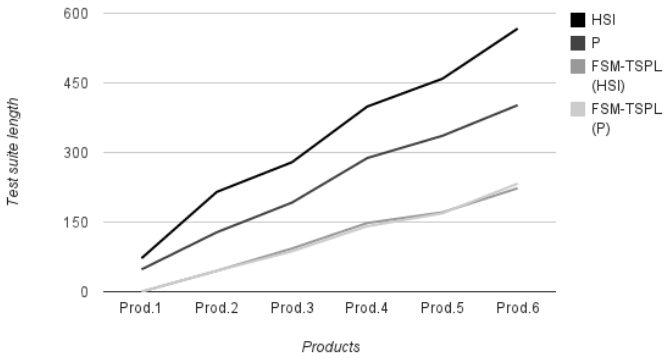


**Figure 7. Cumulative cost for AGM**

The FSM-TSPL (HSI) strategy had saved 61% compared to HSI and 45% compared to the P method. FSM-TSPL(P) saved 59% compared with HSI and 42% compared with P.

### 5.3. Mobile Media

Mobile Media [Figueiredo et al., 2008] is an SPL, based on the MobilePhoto SPL [Young and Young, 2005], that contains several features, such as photo manipulation, music, and videos on mobile devices. This SPL was extended to contain new requirements, alternatives and options. Its feature model is depicted in Figure 8.

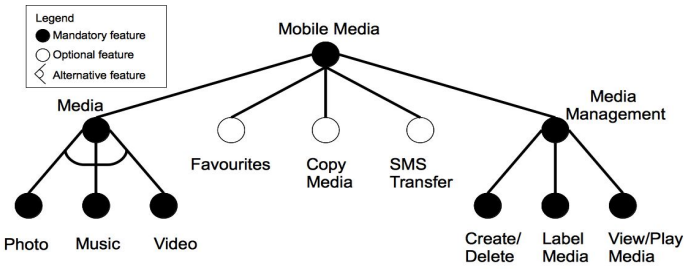


**Figure 8. Feature model for Mobile Media.**

Using Mobile Media, 24 products containing different characteristics were instantiated, so that the products have been instantiated from the Base (mandatory features), and for each product, alternative and optional features were selected. For each product, an FSM was modeled so that the number of states varies from three to six, with eight inputs and two outputs.

The length of test suites for each product is represented in Figure 9. We observe that the test suites generated by the HSI method have the highest costs, while the test suites generated by the FSM-TSPL strategy (both HSI method and P method) have the lowest costs. The P method generated sets at a cost lower than the HSI, but greater than those generated by the FSM-TSPL.

The number of resets is shown in Figure 10. The HSI method also generated the test suite with more reset operations, followed by the P method. The sets generated by the strategy FSM-TSPL have lower number of resets than other methods.

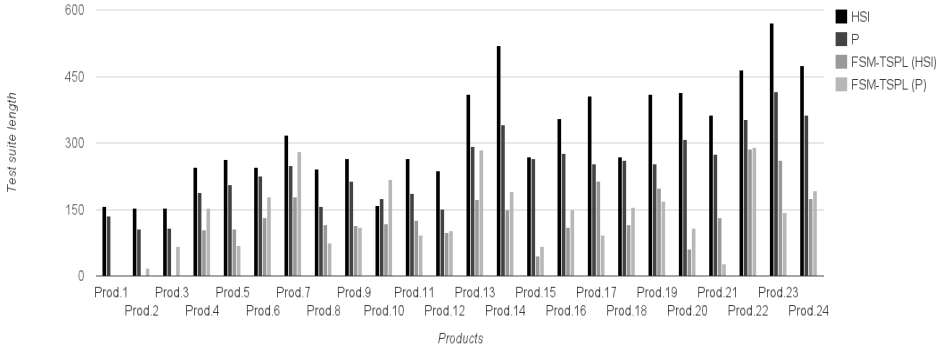


**Figure 9. Test suite length for Mobile Media**

The cumulative cost to test all products in Mobile Media is shown in Figure 11. Notice that the HSI method had the largest cost, the P method has an average cost and similar to HSI while the FSM-TSPL has the lowest cost, so that the FSM-TSPL (HSI) has the overall cost of 3006.

Through the use of the FSM-TSPL (HSI), savings of 61% was achieved compared to HSI and 48% to P. The results showed that the FSM-TSPL (P) strategy had saved 58% compared to HSI and 44% compared to the P method.

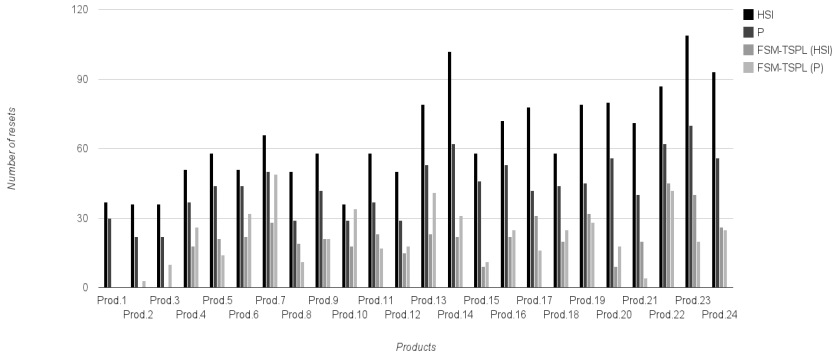


**Figure 10. Number of resets for Mobile Media.**

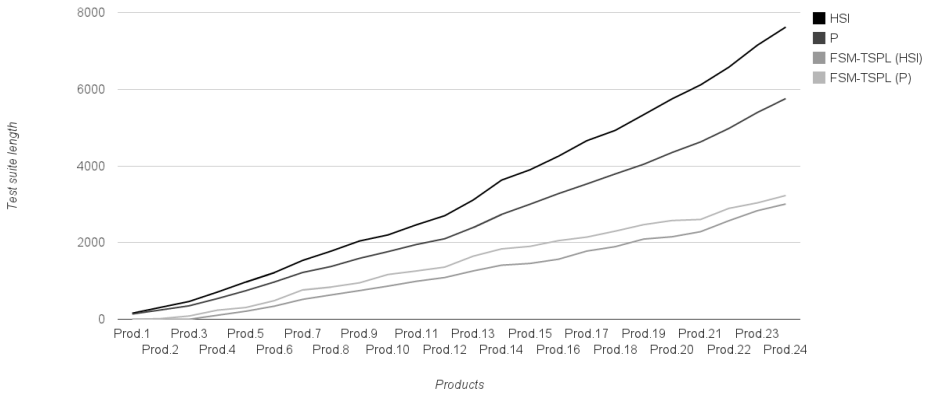


**Figure 11. Cumulative cost for Mobile Media.**

## 6. Conclusions

In this paper, we presented a strategy for *Finite State Machine* (FSM)-based testing of *Software Product Lines* (SPLs). The strategy focused on eliminating the redundant test cases by using the incremental P method. The test suites produced for the products are complete with respect to a fault domain. A case study was conducted with two SPLs, AGM and Mobile Media. Results showed that a considerable effort on test case execution can be saved through the elimination of redundant test cases. The savings of test cases depend on the FSM and test case inputs. In both SPLs, the results were very similar, with a reduction of up to 61% compared with a conservative application of other methods.

In future work, we intend to conduct a detailed experiment using larger SPLs. Moreover, we will develop a tool to integrate the management of SPL artifacts and automate the testing strategy. Further research is also necessary on how cost and effort can be reduced during the test modeling of a product when models exist for different products already tested.

## Acknowledgments

Maykon Capellari was partially supported by the Project PROCAD/CAPES 191/2007 – *"Integrando e aprimorando atividades de pesquisa, ensino/treinamento e transferência tecnológica em teste e validação de software"* and INCT-SEC Adenilso Simao is partially financially supported by CNPq (grant 474152/2010-3). Andre T. Endo is financially supported by FAPESP (grant 2009/01486-9).

## References

Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J.- M. (1999). Pulse: a methodology to develop software product lines. In: symposium on Software reusability (SSR '99), pages 122–131.

Bertolino, A. and Gnesi, S. (2003). Use case-based testing of product lines. In the 9th European software engineering conference (ESEC/FSE-11), pages 355– 358, ACM.

Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., and Pretschner, A. (2005). Model-based testing of reactive systems: advanced lectures. Springer, 1st edition.

Cabral, I., Cohen, M. B., and Rothermel, G. (2010). Improving the testing and testability of software product lines. In SPLC, 2010, pages 241–255.

Chow, T. S. (1978). Testing software design modeled by finite-state machines. IEEE Trans. Softw. Eng., 4:178–187.

Clements, P. and Northrop, L. (2001). Software Product Lines: Practices and Patterns. Addison- Wesley Longman Publishing Co.

Cockburn, A. (2000). Writing Effective Use Cases. Addison-Wesley Professional.

Dalal, S. R., Jain, A., Karunanithi, N., Leaton, J. M., Lott, C. M., Patton, G. C., and Horowitz, B. M. (1999). Model-based testing in practice. In the 21st international conference on Software engineering (ICSE '99), pages 285–294, ACM.

Figueiredo,E.,Cacho,N.,SantA´nna,C.,Monteiro,M.,Kulesza,U.,Garcia,A.,Soares,S.,Ferrari, F., Khan, S., Castor Filho, F., and Dantas, F. (2008). Evolving software product lines with aspects: an empirical study on design stability. In ICSE '08, pages 261–270.

Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M., and Ghedamsi, A. (1991). Test selection based on finite state models. IEEE Trans. Softw. Eng., 17:591–603.

Gill, A. (1962). Introduction to the theory of finite-state machines. McGraw-Hill.

Gomaa, H. (2004). Designing Software Product Lines with UML: From Use Cases to Pattern- Based Software Architectures. Addison Wesley, Redwood City, CA, USA.

Gomaa, H. (2005). Designing software product lines with uml. In SEW ’05: Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop - Tutorial Notes, pages 160–216, Washington, DC, USA. IEEE Computer Society.

Im, K., Im, T., and McGregor, J. D. (2008). Automating test case definition using a domain specific language. Pages 180–185.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute.

Luo, G., Petrenko, A., Petrenko, R., and Bochmann, G. V. (1994). Selecting test sequences for partially-specified nondeterministic finite state machines. In In IFIP 7th International Workshop on Protocol Test Systems, pages 91–106.

MIT, S. D. G. (2010). Alloy.

Olimpiew, E. M. and Gomaa, H. (2009). Reusable model-based testing. In ICSR ’09: Proceedings of the 11th International Conference on Software Reuse, pages 76–85,.

Oster, S., Wubbeke, A., Engels, G., and Schurr, A. (2011). A Survey of Model-Based Software Product Lines Testing, pages 339–381. CRC Press/Taylor & Francis.

Petrenko, A., Yevtushenko, N., Lebedev, A., and Das, A. (1994). Nondeterministic state machines in protocol conformance testing. In Proceedings of the IFIP TC6WG6.1 Sixth International Workshop on Protocol Test systems VI, pages 363–378.

Pohl, K., Böckle, G., and van der Linden, F. J. (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag.

Pohl, K. and Metzger, A. (2006). Software product line testing. Commun. ACM, 49(12):78–81.

SEI (2011). A framework for software product line practice.

Simão, A. and Petrenko, A. (2010). Fault Coverage-Driven Incremental Test Generation. The Computer Journal, 53(9):1508–1522.

Simão, A., Petrenko, A., and Yevtushenko, N. (2009). Generating reduced tests for fsms with extra states. In the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems (TESTCOM ’09), pages 129–145.

Tevanlinna, A., Taina, J., and Kauppinen, R. (2004). Product family testing: a survey. SIGSOFT Softw. Eng. Notes, 29(2):12–12.

Uzuncaova, E., Garcia, D., Khurshid, S., and Batory, D. (2007). A specification-based approach to testing software product lines. In ESEC-FSE ’07, pages 525–528, ACM.

Uzuncaova, E., Garcia, D., Khurshid, S., and Batory, D. (2008). Testing software product lines using incremental test generation. In the 2008 19th International Symposium on Software Reliability Engineering (ISSRE ’08), pages 249–258.

Young, T. J. and Young, T. J. (2005). Using AspectJ to build a software product line for mobile devices. MSc dissertation. In University of British Columbia.

Zander, J., Schieferdecker, I., and Mosterman, P. J. (2011). Model-Based Testing for Embedded Systems. CRC Press.