Criteria and Guidelines to Improve Software Maintainability in Software Product Lines

Gustavo Vale
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais,Brazil
gustavovale@dcc.ufmg.br

Ramon Abílio
IT Department
Federal University of Lavras
Lavras, Minas Gerais, Brasil
ramon.abilio@dgti.ufla.br

André Freire, Heitor Costa
Department of Computer Science
Federal University of Lavras
Lavras, Minas Gerais, Brasil
apfreire@dcc.ufla.br, heitor@dcc.ufla.br

***Abstract*—Software Product Line (SPL) consists of an approach for supporting software design and development to promote large-scale and systematic reuse of components. Reuse in SPLs is enabled by using common features of a domain composing the SPL's core and other features defining variation points. Features can be defined as modules of an application with consistent, well-defined, independent, and combinable functions. Changes in SPLs can be more complex than in single systems because changes in a module can impact on more than one product. In this paper, the goal is to propose criteria for identifying possible problems and guidelines for increasing Maintainability Index (MI) of software generated by SPL. Seven criteria were created and guidelines were elaborated and associated to these criteria. The criteria were used to identify possible problems related to legibility and complexity of features' source code. The guidelines were used to solve or minimize problems identified. To verify the effectiveness of the criteria and guidelines, a case study was performed by applying the criteria and guidelines on TankWar SPL in two conditions: criteria and guidelines applied (i) separately and (ii) cumulatively by following a proposed sequence. After applying criteria and using guidelines, an improvement in the maintainability index of up to 17.65 points was achieved. The study showed that criteria and guidelines were effective to improve the maintainability of TankWar SPL.**

Keywords- Software Quality; Software Maintenance; Software Product Line; Feature Oriented Development

I. INTRODUCTION

Software Product Lines (SPLs) is an approach to design and implement software whose purpose is to promote systematic and large-scale reuse of components [1]. These components can be defined as features (differentiate products and are defined as modules of an application with consistent, well-defined, independent, and combinable functions [2]). Once software is decomposed into features, we can have different versions by (re)combining features [2]. Differences among products are main points to be managed in SPLs (Variabilities) [3]. Managing variability is difficult because of its complexity; to support this management, feature models were created to model SPLs, which can be structured as trees and have different types of features, logical operators, and constraints among unrelated features (called crosstree constraints) [4].

To implement Feature-Oriented SPLs, we can use composition and annotation-based technologies. For example, AHEAD (composition-based technology) whose artifacts that compose common core of SPLs are treated as constants and the others artifacts can be called refinement functions [5]. AHEAD Tools Suite (ATS) was developed [5] to support development with AHEAD and provides tools to support feature implementation and composition.

An industry goal is to develop software with high quality and minimal number of errors, defects, and faults. However, quality is a subjective concept and to assert whether software has high quality is difficult because high-quality software for a developer can be low quality one for a user (vice-versa). For example, user is not preoccupied with internal structure of software; this preoccupation is fundamental for developer. Therefore, measurements must be used to measure software quality, e.g., maintainability [6]. Maintenance in SPL should be more complex that in systems not designed as SPLs, because changes in a module can affect various products. Thus, it deserves attention from academia to provide techniques to help developing more maintainable SPLs.

Number of tools and guidelines to help maintaining or increasing SPL maintainability is limited. This indicates the existence of an open field for research related to SPLs' maintainability. In this paper, criteria and guidelines are described to increase the maintainability of SPL. The main goal is to evaluate the impact of maintenance on the source code of the features that compose a SPL when criteria and guidelines are followed. Guidelines show alternative ways to solve identified problems. To evaluate the effectiveness, a case study was presented by showing variations of the Maintainability Index of products generated from a SPL.

The remainder of this paper is organized as follows. Section II shows concepts on feature-oriented programming, and definition of Maintainability Index. Section III outlines methodological design of the study. Section IV describes proposed criteria and guidelines to improve SPL maintainability. Section V explains development and results of the case study. Section VI summarizes related work. Section VII presents threats to validity. Section VIII discusses conclusion and suggestions of future works.

II. BACKGROUD

In this section, we described theoretical background and terminology necessary to carry out this exploratory study.

A. Feature Oriented Programming and AHEAD

Feature-Oriented Programming (FOP) is used for features modularization and separation. Systems should be systematically constructed by defining and composing of

978-1-4799-8828-0/15 $31.00 © 2015 Crown Copyright
DOI 10.1109/ITNG.2015.75

features [5, 8]. FOP performs synthesis of software into SPLs, in which features are used to distinguish it of the same product family [5]. Therefore, products must be implemented in units of modularization syntactically independent by combining modules that represent features in a flexible way, without loss of resources of static type checking.

One of the technologies to implement FOP concepts is AHEAD [5] (tools that allows refinements in source code and other artifacts). Jakarta language is one of main components of ATS and allows the feature implementation in syntactically independent units (refinements). These refinements allow adding fields and methods into base classes of the base-program and adding extra behavior into existing methods. The main tool of ATS is the composer that receives base application and a set of features as input and generates a base application with the composition of its features as output. Refinements are defined in `jak` files identified by `refines` keyword. In ATS, features are implemented in directories, which contain refinements of classes of the base-program. Thus, composition of features is a composition of directories [5].

B. Maintainability Index

Maintainability Index (MI) was proposed by Oman in 1994 [9] and modified in 1997 [10] by the Software Engineering Institute. MI's goal is to measure quantitatively the system maintainability and to help reducing maintenance costs. To calculate MI, some metrics are used, such as, Halstead's metric and McCabe1's Cyclomatic Complexity. For calculating MI [10]: `MI = 171 - 5.2 x ln (aveV) - 0.23 x aveV(g') - 16.2 x ln (aveLoC) + 50 x sin √2.4 x perCM` where aveV = average Halstead Volume, aveV(g') = average extended cyclomatic complexity, aveLoC = average count of lines of code (LoC), and perCM = average percent of lines of comments.

III. PROCEDURES

The first methodological procedure used was a literature search to identify candidate SPLs developed using AHEAD. We chose TankWar SPL because it was reported in other studies [4, 11]. A detailed analysis of the source code was performed for better understanding of TankWar SPL. After, we obtained number of lines of code for each method and number of lines of comments, for example. We studied coding standards, structural problems of source code, and refactoring methods. Legibility and complexity problems were pointed out (criteria) and ways to solve/minimize them were described (guidelines). Next, we decided to use MI due to its wide use in industry and academia, including in the context of SPLs [12].

The next step was to determinate which products generated from the SPL would be analyzed and modified using the proposed guidelines. We wanted products that involved all SPL features. The SPL has two platforms (PC and Handy); this means that one product can run only in one platform; it is not possible to use all features in the same product. Therefore, we decided to choose three products with minimum, intermediate, and maximum number of features for each platform (six products). We believed that all features were included and had products with different numbers of lines.

Subsequently, guidelines were used in two phases: i) legacy SPL was replicated in 7 different projects and, in each replication, each problem-identification criterion was evaluated and respective guidelines related to the criterion were used; and ii) criteria were evaluated and guidelines were followed cumulatively for the same SPL. For each guideline used, we tested products to analyze whether side effects were propagated. When a side effect was detected, refactoring was undone and redone or an alternative way was sought for its application. After, MI was calculated and analyzed, and comparisons and discussions were done.

We performed quantitative analyses of the differences in MI of the products before and after using guidelines. We used Related-Samples Signed-Rank Wilcoxon tests to verify whether there were significant differences in MI using a 5% significance level.

IV. CRITERIA AND GUIDELINES

Definition of criteria to identify maintainability problems and elaboration of guidelines for improving them are needed to guide software engineers and developers to perform maintenance and increase the SPL's maintainability. Criteria proposed in this paper represent potential problems and elaborated guidelines help solving/minimizing the problems identified. These criteria can be applied in development or maintenance of SPL. Each criterion has description, motivation, and set of guidelines to be followed (Table I). The only specific criterion to AHEAD is **Criterion 4**, but its associated guidelines can be applied to other technologies using comments to express whether a method is new or an override in a refinement. These criteria were defined and associated guidelines were designed to evaluate and maintain the source code of SPL. If a criterion is not completely addressed, guidelines related to this criterion should be used to modify the source code.

I. CASE STUDY

TankWar SPL is a game developed by students at German University of Magdeburg as a SPL to adhere to portability requirements common to mobile devices [11]. It is a SPL of ~5,000 LoC and composed of 37 features [4]. Moreover, it can generate software products for personal computers (PCs) and mobile devices (*Handy* in German). TankWar SPL offers possibility of generating 39,060 different products, but we chose six products for analyzing. These products have different representative factors of the SPL variability (products with minimum, intermediate, and maximum number of features to PC and Handy platform). We believed that these products can express a representative set of functionalities of TankWar SPL, because all features are used. Intermediate products were chosen using the average of the number of features and the features with a larger LoC were chosen to compose these products.

TABLE I. CRITERIA: DESCRIPTION, MOTIVATION, AND GUIDELINES

ID	Criteria	
1	**Description**	Is the goal of the refinement explicit in the source code?
	Motivation	The presence of documentation comments indicating the purpose of the implementation of refinement facilitates the code understanding/analysis.
	Guidelines	1 - Add at the beginning of class/refinement/method a comment stating the purpose of their implementation.
		2 - Rewrite the comment if there is one, but this one is not the goal of the class/refinement/method, although it is explicit what the code does.
2	**Description**	Do the methods of a class have too much responsibility?
	Motivation	A method that has a lot of responsibility is difficult to refine.
	Guidelines	1 - Find methods amenable to refactoring (can be used the number of lines of code and/or cyclomatic complexity per method).
		2 - Refactor the method met to make it more concise using the refactoring methods: *Extract Method, Replace Temp with Query* and *Decompose Conditional*.
3	**Description**	Are there clones of source code at the features implemented?
	Motivation	Code clones can cause negative effects on software system reducing the maintainability and introducing errors.
	Guidelines	1 - Find code clones and check the reason for its existence (e. g., failure of programing or limitation of technology).
		2 - Refactor the code to eliminate the clones, when it is possible; using the refactoring methods: *Extract Method, Pull Up Field* and *Form Template Method* [7].
4	**Description**	Are the mechanisms *new* and *overrides* used to indicate explicitly *new* and *overridden* methods?
	Motivation	In the chain of refinements, a method can be refined several times and new methods can be added by refinements. The explicit indications of *new* and *overridden* methods assist in maintaining and indicate to programmer pay attention with present methods in the chain of refinement.
	Guidelines	1 - Find new methods and add new on the declaration of this method.
		2 - Find overridden methods and add overrides on the declaration of this method.
5	**Description**	Do the classes of a feature have too much responsibility?
	Motivation	Excessive responsibility in a class can indicate the existence of God Class. This type of class can generate problems on the chain of refinements on the evolution of SPL, because it extends the possibility of undue refinements which increase the amount of responsibility of class. In addition, a class with methods and attributes unduly public or protected can facilitate the coupling.
	Guidelines	1 - Find Classes with too much responsibility (e. g., excessive methods and/or attributes public and protected).
		2 - Refactor classes using the refactoring method: *Extract Class* [7].
6	**Description**	Are the data of class encapsulated?
	Motivation	In the chain of refinements may be necessary to access attributes to access their values. The *mixin* composition engine composes the features using a sequence of inheritance, so the attributes are visible to subclasses. It should implement the encapsulation, keeping private attributes and proving accessor methods to reduce the coupling.
	Guidelines	1 - Find the classes that have public or protected attributes.
		2 - Refactor classes using the refactoring methods: *Encapsulate Field, Remove Setting Method, Move Method* and *Hide Method* [7].
7	**Description**	Is the writing of code following the Java Code Conventions?
	Motivation	Code conventions improve the reading of source code facilitating software engineer to understand the code.
	Guidelines	1 - Inspect the source code of the features adapting it to Java Code Conventions.

In spite of TankWar SPL having 31 concrete features (and 6 abstract features), product with maximum number of features has 23 features and product with minimum number has 6 features. This fact is due to the crosstree constraints, logical operators, and number of mandatory features. The chosen products have 1,838; 1,802; 3,051; 3,917; 3,330; 3,476 of lines of code for products namely `Product1MinimumPC` (P1; MI = 81.02), `Product2MinimumHandy` (P2; MI = 81.27), `Product3IntermediatePC` (P3; MI = 79.08), `Product4IntermediateHandy` (P4; MI = 77.79), `Product5MaximumPC` (P5; MI = 79.67), and `Product6MaximumHandy` (P6; MI = 78.41).

The problem-identification criteria and corresponding guidelines were applied in two phases. In **first phase**, we replicated the SPL to apply each criterion individually and separately, and applied one criterion for each replica (copy) (Table II). After, we verified what the impact of each criterion on chosen product is. When comparing MI in Table II with MI of the products from Legacy TankWar SPL, three types of results were obtained: positive (when MI increased), neutral (when MI remained the same), and negative (when MI decreased). These values were obtained by the difference between MI of the products after using the set of guidelines and MI of the products of Legacy TankWar SPL (Table III).

We performed a set of Related-Samples Wilcoxon Signed Rank tests (using R statistical software) to verify whether there was significant difference between MI of the products after applying each criterion. Significant differences between MI of the products before and after applying the criteria were found: i) **Criterion 1** (V = 0, N = 6, p-value = 0.031), **Criterion 2** (V = 21, N = 6, p-value = 0.031), **Criterion 3** (V = 21, N = 6, p-value = 0.031), **Criterion 5** (V = 0, N = 6, p-value = 0.031), **Criterion 6** (V = 21, N = 6, p-value = 0.031), and **Criterion 7** (V = 21, N = 6, p-value = 0.031). No significant difference was found after applying **Criterion 4**. V is test result, N is sample size (6 products), and p-value is significance level.

TABLE II. MI AFTER USING GUIDELINES

	Products					
Criterion	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
1	99.58	95,83	93.27	92.26	91.20	90.24
2	81.01	81.20	78.85	77.54	79.46	78.18
3	80.29	80.29	78.50	76.90	79.08	77.60
4	81.02	81.27	79.08	77.79	79.67	78.41
5	81.46	81.61	79.40	78.05	79.99	78.67
6	78.00	78.27	77.17	76.05	77.89	76.78
7	79.50	79.40	76.91	75.36	77.55	76.05

TABLE III. MI VARIATION AFTER USING GUIDELINES FOR PRODUCTS RELATED TO LEGACY TANKWAR SPL

	Products					
Criterion	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
1	18.56	14.56	14.19	14.47	11.53	11.83
2	-0.01	-0.07	-0.23	-0.25	-0.21	-0.23
3	-0.73	-0.98	-0.58	-0.89	-0.59	-0.81
4	0	0	0	0	0	0
5	0.44	0.34	0.32	0.26	0.32	0.26
6	-3.02	-3.00	-1.91	-1.74	-1.78	-1.63
7	-1.52	-1.87	-2.17	-2.43	-2.12	-2.36

In spite of the negative results obtained after applying guidelines related to four criteria (**Criterion 2**, **Criterion 3**, **Criterion 6**, and **Criterion 7**), we believed that some alterations have beneficial effects not portrayed at MI. For example, following Java Code Conventions, we introduced new lines to source code, because it states that declarations of variables need be one for each line. These lines have caused alterations on MI. In others cases, we believed that

these results happened for products and not to SPL as a whole. For example, when `To Up Field` method is used, one transformation on feature model occurs (one abstract feature is transformed in a concrete feature). With this transformation, some code lines need go to a new class (constant) for reducing number of lines and removing code clone of the SPL. However, some methods need to be declared in the constant and some functionality is added on refinement. This method in SPL context is better explained in another paper [11]. Thus, number of lines can be larger for one product, but can be smaller for the SPL.

As result of the first phase, we can observe how to determine an order wherein the criteria could be applied to make the maintenance process more efficient and reduce rework. Much of the rework that occurs is due to some refactored methods used in more than one problem-identification criteria. Besides, when legibility criteria are applied first, the effort can be reduced because refactored methods require larger and more complex variations than legibility and organization criteria on source code. Thus, we have determined a optimized order: **Criterion 7**, **Criterion 1**, **Criterion 4**, **Criterion 2**, **Criterion 3**, **Criterion 6**, and **Criterion 5**. We have presented the rationale for the application of each criterion in the proposed order:

- **Criterion 7**: establish a default encoding can prevent rework for use the guidelines of others criteria;
- **Criterion 1**: document classes and methods can help understanding source code and refactoring methods;
- **Criterion 4**: make refinements of methods more explicit calls the attention of programmers, if any refactoring happens in these refinements;
- **Criterion 2**: refactor long methods can help in class refactoring and code clone refactoring;
- **Criterion 3**: reduce code clones can avoid rework on refactoring classes;
- **Criterion 6**: have a class with encapsulated data can facilitate the modification of classes;
- **Criterion 5**: know the code and having the data encapsulated is important in the process of refactoring of a class because it is a more complex task and, therefore, should be performed at the end.

The **second phase** had the goal to determine the effect of applying the guidelines related to criteria when used cumulatively in the proposed order on the sample of six products of the same SPL. MI after performing each step is shown in Table IV. For example, after applying the **Criterion 7**, **Criterion 1**, and **Criterion 4** for P3, MI was 93.16. The SPL with 7 criteria applied was called New TankWar SPL. We compared the impact of each criterion when applied in a cumulative way. For example, when **Criterion 7** was applied on Legacy SPL the resulting MI was the same as when applying this criterion separately (in first phase). Table V shows the variation on MI of each product comparing MI after applying one criterion and the previous MI of the product immediately before its application in a cumulative fashion.

If the criteria in Table V were sorted on crescent order (Table III) and compared, we can compare the increments of MI of the criteria when applied separately and when applied cumulatively. Table VI presents difference between increments obtained by applying the criteria cumulatively and separately (cumulative minus separate).

TABLE IV. MI AFTER USING CUMULATIVE GUIDELINES

	Products					
Criteria	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
7	79.50	79.40	76.91	75.35	77.55	76.05
7, 1	99.83	96.70	93.16	92.13	91.55	90.55
7, 1, 4	99.83	96.70	93.16	92.13	91.55	90.55
7, 1, 4, 2	99.81	96.45	93.19	92.14	91.49	90.48
7, 1, 4, 2, 3	99.26	95.55	92.96	91.82	91.13	89.93
7, 1, 4, 2, 3, 6	97.01	92.65	91.99	90.93	90.06	89.09
7, 1, 4, 2, 3, 6, 5	98.67	95.02	93.01	91.95	91.23	90.26

TABLE V. MI VARIATION USING CUMULATIVE GUIDELINES

	Products					
Criteria	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
7	-1.52	-1.87	-2.17	-2.43	-2.12	-2.36
7, 1	20.33	17.30	16.25	16.78	14.00	14.50
7, 1, 4	0	0	0	0	0	0
7, 1, 4, 2	-0.02	-0.25	0.03	0.01	-0.06	-0.07
7, 1, 4, 2, 3	-0.55	-0.90	-0.23	-0.32	-0.36	-0.55
7, 1, 4, 2, 3, 6	-2.25	-2.90	-0.97	-0.89	-1.07	-0.84
7, 1, 4, 2, 3, 6, 5	1.66	2.37	1.02	1.02	1.17	1.17

TABLE VI. DIFFERENCE IN MI INCREMENTS USING GUIDELINES CUMULATIVELY IN RELATED TO APPLY THEM SEPARATELY

	Products					
Criterion	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**
1	1.77	2.74	2.06	2.31	2.47	2.67
2	-0.01	-0.18	0.26	0.26	0.15	0.16
3	0.18	0.08	0.35	0.57	0.23	0.26
4	0	0	0	0	0	0
5	1.22	2.03	0.70	0.76	0.85	0.91
6	0.77	0.10	0.94	0.85	0.71	0.79
7	0	0	0	0	0	0

As well as for the analyses on the effect of using the criteria separately, we performed analyses to verify if there was significant difference between applying each criterion separately and applying the criteria in the proposed order. We compared the variation on MI of the products immediately before and immediately after applying the criteria in the proposed sequence and the variation obtained when the same criterion was applied separately. we used a set of Related-Samples Wilcoxon Signed Rank tests to perform the comparisons. Significant differences were found in variations of MI when the criteria were applied in the proposed sequence for **Criterion 1** (V = 0, N = 6, p-value = 0.031), **Criterion 3** (V = 0, N = 6, p-value = 0.031), **Criterion 5** (V = 0, N = 6, p-value = 0.031), and **Criterion 6** (V = 0, N = 6, p-value = 0.031). No significant differences were found for **Criterion 2**, **Criterion 4**, and **Criterion 7**. V is test result, N is sample size (6 products), and p-value is significance level

After changing, New TankWar SPL has a total of 7,450 lines, including source code (59.6%), documentation (19.6%), comments (0.3%), and blank lines (20.5%) (Table VII). It is composed of 92 classes and three continued to be of interface type. Number of refinements of classes and interfaces continued proportionally equivalent (68% of classes) with changes from 53 to 63. Number of methods was changed from 308 to 354, where 139 (39.26%) of them belong to refinements of class. Number of imports was changed to a lesser scale from 87 to 88. This change

happened because of the modifications caused by using guidelines from **Criterion 3** (removal of cloned code). Number of attributes was changed from 236 to 208, because of using guidelines from **Criterion 3**, when the code clones were changed to classes from features hierarchically superior on feature model and some attributes were removed. This fact also happened with two constants removed; they were moved from `Maler` class of `PC` and `Handy` features to `Maler` class of `Platform` feature. In comparison with the attributes of the access modifier, New TankWar SPL had a higher degree of encapsulation, because of number of default modifiers reduced from 52 to 16, public one from 20 to 16, and protected one from 128 to 84, but private one increased from 36 to 110.

TABLE VII. VALUES OF MEASUREMENTS USED TO MAP LEGACY AND NEW TANLWAR SPL

Measures	SPL	
	Legacy	New
Number of Total of Lines	5,640	7,450
Number of Lines of Code	4,865	4,442
Number of Lines of Documentation Comments	44	1,460
Number of Lines of Comments	61	22
Number of Classes (Constants)	29	33
Number of Interfaces (Constants)	1	1
Number of Refinements of Classes	56	61
Number of Refinements of Interfaces	2	2
Number of Methods	308	354
Number of Methods on Refinements of Class	97	139
Imports	87	88
Attributes	236	208
Constants	53	51

When comparing MI of the New TankWar SPL and MI of the Legacy TankWar SPL, increase of the maintainability is notable. We cannot calculate the MI of one SPL, aggregating MI variation of the 7 criteria when the criteria are applied separately. Thus, comparison presented in this paper is only with MI of the six products of Legacy TankWar SPL and six products of New TankWar SPL. The greatest difference of MI was 17.65 points in P1 and the lowest difference was 11.56 points in P5. The average MI variation was 13.82 points. We can observer that results were positive in all products, especially by the perception of deficiency of documentation of Legacy TankWar SPL. When the guidelines were used in a cumulative way, new methods have become documented and the results were better. This may show the importance and the influence of documentation of source code in MI. With well-documented code, it helps enhancing the code's understanding and makes its maintenance a less arduous.

II. RELATED WORK

One way to measure the maintainability of SPLs is described in a study [12] using MI. The authors argued that an approach that involved calculating MI for each feature separately and then aggregating those values to obtain MI of the product could lead to wrong results. They devised an indirect way to measure feature MI. In the approach proposed, authors obtained all possible products from SPL could generate and calculated two more values - one referring to products exhibiting one feature and other products that lacked the feature. The results were formalized in terms of a matrix with MI contribution of each feature.

A set of procedures and guidelines were proposed to evaluate systems quality [13]. This set consists of a checklist that assesses quality characteristics defined on ISO/IEC 9126. To evaluate the maintainability, it required a scheme in which evaluation of initial version and modified version of a product are considered. After becoming acquainted to initial version and changes on the software to be performed by the developer in a limited time, system maintainability was estimated. The time must be set according to number of recommended changes and the requirements of the current market the system are intended for. These modifications were called MEDE-PROS.

A set of criteria to assist in maintainability was proposed [14] in order to create/evaluate methodologies for the development of maintainable software. The criteria were divided into two categories: i) primary, to help determining the ease of maintenance during the system's lifetime, especially for adaptive maintenance; and ii) secondary, to assist in determining the quality of the produced system, which helps complying with the requirements of users, in particular referring to corrective and perfective maintenance. The criteria were used to evaluate five methodologies for system development.

A set of guidelines was created by Etxeberria *et al*. [15] with the objective of maintaining the quality consciously on SPL engineering. In this study, five key points to keep the SPL quality were considered: i) design of variability quality; ii) design quality; iii) architecture evaluation; iv) implementation quality; and v) test quality. After, 16 methods were presented and divided by properties, such as goal, type of attributes, and evaluation techniques. Although several methods are available, authors concluded that there is a gap among approaches, making necessary to perform more studies on the SPL subject.

This study differs from others because the maintainability evaluation happened on the source code from the features of AHEAD SPL. We changed the focus to determining the SPL maintainability to measure the maintainability for the main products generated by SPL. We believed that this approach is a simpler way to estimate the SPL maintainability, as other approaches such as defined by Aldekoa *et al*. [12] to measure MI of SPL can arduous and complicated.

III. THREATS TO VALIDITY

We need highlighting that many decisions were taken to find the results of this work. First, we chose a metric to assess the maintainability, the MI was chosen because we found another work that used this metric and presented a good way to calculate the SPL maintainability. However, we assumed that this could not be the most adequate metric. It is possible that we may have some error calculating the MI and the Wilcoxon tests. To try to minimize it, two researchers computed the metrics and calculating MI in different computers. If some anomaly was found all process of measuring or statistical analyzing should start again. In despite of, we used as standard 5% significance level to run the Signed-Rank Wilcoxon tests. We assumed that another percentage could be chose and different results can be obtained. The gap of a tool to measure the maintainability of

AHEAD-SPLs (one limitation) did that we chose some products, in general enterprises maintenances a reduced number of products, but the maintainability should be measured from the SPL or from the SPL features. We believed that we chose products that represent TankWar SPL, but it is difficult to affirm it. The criteria were defined only with the problems detected in the SPL studied, although treating generic problems. We can say that some of these criteria are going to be found in others SPLs, but results can be different from ours.

IV. CONCLUSIONS

Software development must occur ever faster to meet market demand. These systems have little variation and can be developed by SPLs using features. The benefits of building systems using SPLs go beyond the rapid development, because it can promote systematic and large-scale reuse of components. If performed properly, this reuse can increase systems quality and maintainability. Among the technologies available for the implementation of SPLs, we chose AHEAD, a feature-oriented technology that seems usual and have been used in others works. Features allow differentiating products and can be defined as modules of an application with consistent, well-defined, independents, and combinable functions [2].

The main goal of this work was to evaluate MI of software derived from SPL. The evaluation was conducted based on 6 products generated from TankWar SPL. With that SPL, we can derive around 40,000 products, but six products were chosen to use a representative set of features from TankWar SPL. The evaluation was conducted as follows. Based on the seven criteria defined, we performed an analysis and, when a criterion was not met, a set of associated guidelines was used to try increase MI. Comparisons to identify the improved maintenance of systems were made on MI before and after using guidelines. The use of guideline was done in two phases: i) guidelines of each criterion were used in separate way on 7 replicas from Legacy TankWar SPL; ii) guidelines were used cumulatively in a proposed order determined to enhance its use. The results were satisfactory and MI on second phase increased by 17.65, 13.75, 13.93, 14.16, 11.56, and 11.85 points on 6 products analyzed. These values were obtained by the difference between MI from New TankWar SPL and MI from Legacy TankWar SPL. The statistical analyzes by Signed-Rank Wilcoxon tests provides that 6 criteria provide significant differences in MI.

The result was positive using guidelines because some shortcomings of Legacy TankWar SPL were remedied, such as, lack of documentation comments (**Criterion 1**). When considering **Criterion 1**, an analysis of the quality of the comments should be performed because software with many characters and/or lines of comments does not necessarily make it easier to understand if these comments do not report the objective of class, methods or their refinements.

In the analysis of the application of the guidelines separately, we observed that MI was reduced when we used guidelines of **Criterion 2**, **Criterion 3**, **Criterion 6**, and **Criterion 7**, because the evaluation happened into the products and the changes happened into the SPL features. In some cases, such as clone reductions (**Criterion 3**), number of lines of code was reduced for the SPL and increased for the individual products, because of classes and refinements of classes and existing methods. We concluded that the products' MI does not vary directly and proportionally with MI by the TankWar SPL. Furthermore, performing maintenance on SPL is not as trivial as changing single systems, because changes in several modules can generate side effects in many products and tests should occur in several products and not only in one product.

Therefore, main contribution of this work corresponds to definition, testing, and evaluation of a set of criteria and guidelines. The testing and evaluating were performed on TankWar SPL and results showed the criteria and guidelines were effective to help increasing maintainability of SPLs. In addition, an order was suggested to avoid rework and make the maintenance process more efficient. As limitations, we did not find any tool to measure the SPL maintainability or able to calculate some metric from AHEAD-SPLs; and, this work is limited to treat only anomalies that aim increase legibility and reduce the complexity.

Some suggestion for future work include developing a tool for measuring the SPL maintainability developed by technologies based on composition, adding other measures to assess the maintainability of SPL, defining other criteria and guidelines to different points in relation to Maintainability.

ACKNOWLEDGMENT

We thank FAPEMIG, CNPq e CAPES by financial support.

REFERENCES

[1] P. Clements; L. M. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley. Vol. 3, 563p., 2002.

[2] Q. Boucher; et al. Introducing TVL, a Text-Based Feature Modelling Language. International Workshop on Variability Modelling of Software-intensive Systems, pp. 159-162, 2010.

[3] P. Y. Schobbens; et al. Feature Diagrams: A Survey and a Formal Semantics. International Requirements Engineering Conference, pp. 139-148, 2006.

[4] S. Schulze; et al. Variant-Preserving Refactoring in Feature-Oriented Software Product Lines. Workshop on Variability Modeling of Software-Intensive Systems. pp. 73-81, 2012.

[5] D. Batory, Feature-Oriented Programming and the AHEAD Tool Suite. International Conference on Software Engineering, pp.702-703, 2004.

[6] ISO/IEC 25000. Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SquaRE. 2005

[7] M. Fowler; et al. Refactoring: Improving the Design of Existing Code. Addison-Wesley, ISBN: 0-201-485672, 464p., 1999.

[8] J. Liu; D. Batory; C. Lengauer, Feature Oriented Refactoring of Legacy Applications. ICSE, pp. 112-121, 2006.

[9] P. W. Oman; J. R. Hagemeister, Construction and Testing of Polynomials Predicting Software Maintainability. Journal of Systems and Software, pp. 251-266, 1994.

[10] SEI - Software Engineering Institute. Software Technology Reference Guide - A Prototype. Carnegie Mellon University Pittsburg, 436p. 1997.

[11] S. Schulze; et al. Code Clones in Feature-Oriented Software Product Lines, International Conference on Generative Programming and Component Engineering, pp. 103-112, 2010.

[12] G. Aldekoa; S. Trujillo; G. Sagardui; O. Díaz, Quantifying Maintainability in Feature Oriented Product Lines. CSMR. pp. 243-247, 2008.

[13] M. T. V. Aguayo; et al. Avaliação da Manutenibilidade de Produtos de Software. IADIS Ibero-Americana WWW/Internet. pp. 432-436, 2005.

[14] A. Deraman; P. J. Layzell, Software Design Criteria for Maintainability. Journal of Science and Technology, volume 3, pp. 1-18, 1995.

[15] L. Etxeberria; et al. Quality Aware Software Product Line Engineering. Journal of the Brazilian Computer Society, volume 14, pp. 57-69, 2008.