Article

Semantics of the maturity model for feature oriented domain analysis

M. Javed[1], **M. Naeem**[1], **H. A. Wahab**[2]

[1]Department of Information Technology, Hazara University, Mansehra, Pakistan

[2]Department of Mathematics, Hazara University, Mansehra, Pakistan

E-mail: mjavedgohar@hotmail.com, naeem@hu.edu.pk, wahabmaths@yahoo.com

Received 6 July 2014; *Accepted 10 August 2014*; *Published online 1 March 2015*

Abstract

Assessing the quality of a model has always been a challenge for researchers in academia and industry. The quality of a feature model is a prime factor in software development because it is used in the development of products. This paper elaborates on our previous work where, we have motivated the need of the maturity model along with the description of such model for feature oriented domain analysis. Here, we provide the semantics of such maturity model. Furthermore, in this extended version, we present an algorithmic technique for the detection of quality level for a given feature model.

Keywords quality of feature models; maturity model; errors; inconsistencies; dead features; invalid feature model.

Computational Ecology and Software
ISSN 2220-721X
URL: http://www.iaees.org/publications/journals/ces/online-version.asp
RSS: http://www.iaees.org/publications/journals/ces/rss.xml
E-mail: ces@iaees.org
Editor-in-Chief: WenJun Zhang
Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction

Producing things in large amount require standardized processes, especially for the similar products. Companies are organizing their production in large amount of production (Benavides, 2010). To reuse existing systems in a systematic way, service-oriented systems resemble supply chain where products manufactured from supplied parts. Same case is for complex service-oriented systems, which needs third party services (Thomas, 2008). For example, car producer offer variation on a model with variable engines, gearboxes, audio and entertainment systems. Example of software services is online travel agency, which may use third-party services for hotel booking, invoicing and for payment option (Naeem, 2012). Similarly, increasing number of software systems with almost similar requirements guide us to Software Product Line (SPL) (Böckle, 2005). SPL Engineering helps in the development within application domain by considering their commonalities and variability. In SPL approach, products are being created by reusability (Clements and Linda, 2002).

SPL incorporating the property of similarities and variability in the family of software is a new technique in the development of software. This helps in the development of high quality software in a short period of time with low budget. Progress has been improved in the development by adopting SPL (Mendonça, 1999).

Features represent the aspects of these software (Kang et al., 1990). To get a valid combination of these features we use feature model which depicts the relationships of these features and constraints on them (Batory, 2005). Usually feature models are tree like structures describing successive refinement of the variability in a product-line. Feature models were proposed back in 1990 as feature oriented domain analysis (FODA) (Kang et al., 1990).

The use of high quality process ensures the good quality resulting products. Hence, it is very important to investigate the quality of the selected model before putting it into practice. In other words, one can say that the quality of a feature model has prime importance because it contributes towards the development of high quality products. There are number of properties which affect the quality of a feature model. One of the agreed deficiencies in feature models is errors in the feature model.

The quality of a feature model can be analyzed from different perspectives which may includes: how efficiently it captures a given domain by keeping the integrity of model itself. The lesser are the occurrences of redundancies, anomalies and inconsistencies in a feature model, the more will be the integrity of a feature model (Maßen and Horst, 2004; Rosso, 2006; Javed et al., 2014).

In our work in (Javed et al., 2014), we presented the first step for the development of the framework to judge the quality of a given feature model, which we call Maturity Model for FODA. In the current paper, we present the semantics of that maturity model. These semantics are based on algorithms. These algorithms get input the feature model and provide the quality level after judging the errors existing in that feature model.

The rest of the paper is arranged as follows: Sections 2 and 3 consist of the discussion of background and related work, respectively. Section 4 provides semantics of the proposed model. Section 5 concludes the paper and highlights the future directions independently. It allows integrating of differential equations and deducing of model to system of recurrence equations.

2 Background

Feature models were introduced by Kang in the form of technical report on FODA in 1990. A feature is prominent characteristic of a product (Kang et al., 1990). Feature model is a hierarchical model that captures the commonality and variability of SPL. The set of permissible selection of features from a feature model is called an instance (Rosso, 2006). Semantics of a valid instance may include: 1) If a feature is chosen then its mandatory feature must be selected in that instance; 2) If a feature is selected in an instance then its optional sub-features can be selected or rejected depending on the preferences; 3) if a feature is selected then exactly one feature from its alternative group must also be selected; 4) Is a feature is selected then at least one feature from its Or-group must also be selected.

Apart from these constraints, a feature diagram may have: Requires constraint) If a source of requires constraint is selected that its target must also be chosen in that instance; Excludes Constraint) Source and target features of excludes constraint cannot be selected in an instance. Thus $\{MP, Ca, Sc, Ba\}$ is the valid instance of a feature diagram shown in Fig. 1. More formally: a feature model and its instance can be defined as:

Definition 1 (Feature Model and Instance – adapted from Rubin and Chechik (2012)):

Given a universe of elements $\mathbb{F}$ *that represent features, a feature model* $\mathcal{FM} = \ll \mathcal{F}, \phi \gg$ *is a set of features* $\mathcal{F} \epsilon 2^{\mathbb{F}}$ *and a propositional formula* ϕ *defined over the features from* $\mathcal{F}$*. An instance* $\mathcal{Ist}$ *of* $\mathcal{FM}$ *is a set of selected features from* $\mathcal{F}$ *that respect* ϕ *(i.e.,* ϕ *evaluates to true when each variable* f *of* ϕ *is substituted by true if* $f \epsilon \mathcal{Ist}$ *and by false otherwise.)*

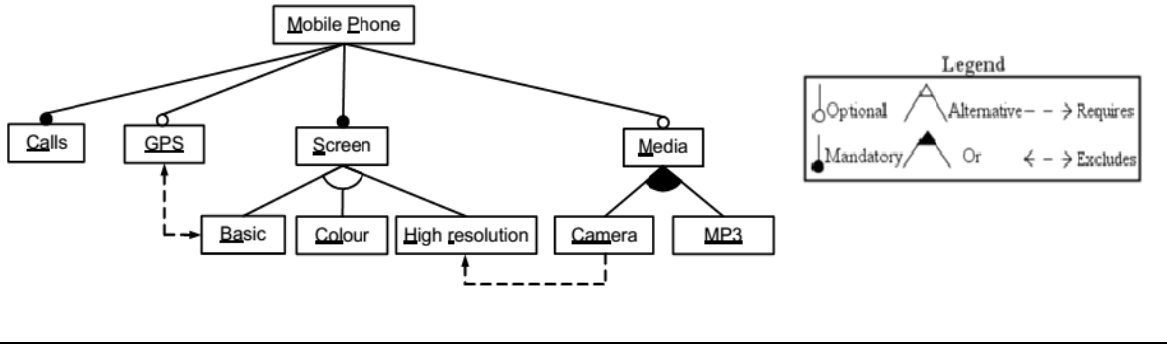

Fig. 1 A feature model of a mobile phone (Benavides, 2010)

On the basis of above definition the propositional formula ϕ of feature model $\mathcal{FM}$ shown in Fig. 1 can be stated as:

$$"(MP \leftrightarrow Ca) \ \wedge (GPS \rightarrow MP) \ \wedge \sim(GPS \wedge Ba) \wedge ((Ba \leftrightarrow Sc) \wedge \sim(Col \vee Hr)) \wedge ((Col \leftrightarrow Sc) \wedge \sim(Ba \vee Hr)) \wedge ((Hr \leftrightarrow Sc) \wedge \sim(Ba \vee Col)) \wedge (Me \rightarrow MP) \wedge (Me \leftrightarrow (Cam \vee MP3)) \wedge (Cam \rightarrow Hr)"$$

Valuations for which this formula is true characterise the valid instances. Here, a possible instance is the valuation that assigns true to $\{\text{MP, Ca, Sc, Hr, Me, Cam}\}$ and false to $\{\text{MP, Ca, Sc, Hr, Me, Cam}\}$.

Feature models may also suffer from errors. In most cases these errors are caused due the wrong use of constraints. For example, use of excludes constraints between mandatory features of a feature model etc. To keep the attention of reader on semantics we have not added that discussion here, we would recommend the interested readers to have a look to Javed et al. (2014) for the detailed discussion.

3 Related Work

To the best of our knowledge there is no attempt in the literature to propose a maturity model that can judge the quality of a given feature model. The only attempt in the literature is ours in which we have motivated the need of such maturity model for feature oriented domain analysis. Although, there are some efforts to propose quality models for feature models but those models are not based on errors in feature models. For example, Thörn (2010) has provided a quality model for evaluating feature models (FMQ). There model is based on the quality factors, quality attributes, indicators and metrics related to feature models and their development.

Ahmed and Fernando in (2011) have provided a business maturity model of software product lines. The motivation of their attempt was to create a strategy for the assessment of the business elements of software product line process. Again, we focus on the existence of errors in feature models and judge the quality of a feature model. Our framework will help designers of software product lines to judge the level of individual feature models rather than the whole process of software product line engineering.

4 Semantics of Maturity Model

Semantics of the maturity model are based on algorithms that will be used to judge the quality level of a given feature model. Our algorithms are divided into three groups: 1) sets the basic attributes value; 2) detects contradiction; 3) finds the selectable and non-selectable features from a feature model. Each algorithm will be discussed in three steps:

Step 1. *Algorithm:* It contains the pseudo code of the algorithmic logic.

Step 2. *Explanation:* In this step every statement is explained in a tabular form with statement number in the first column whiles the explanation in the second column.

Step 3. *Tracing:* It shows the application of algorithm on example.

5 Setting Feature's Attributes

Setting attributes algorithm is the prerequisite of all the upcoming algorithms, so this should be executed before the execution of all other algorithms. The selection and rejection of features in a feature model is based on values of these attributes of features. Without this algorithm no further processing can be done.

5.1 Set attributes algorithm

a) Algorithm

```
1   foreach  f  in  FM
2       If  f.selected ≠ true  OR  f.selected ≠ false
3           f.selected ← null
4           f.relevance ← mandatory / optional / alternative / Or /null
5           f.parent ← this.parent
6           f.exclude[] ←  // names of features that are excluded by this feature or that exclude this feature
7                   f.requires[] ←  // names of features that are required by this feature by implies
8           f.required_by[] ←  //names of features that require this feature by implies
9                   if f.parent ≠ null  then  f.parent.total_alternstesets ←  0, 1, 2 ….
10          if f.relevance = alternate  AND f.selected ≠ false
11                  alternateset ← a/b/c …
12                  f.parent_alternateset.add(f)
13                  End of if condition started at line # 11
14          End of if condition started at line # 02
15  End of foreach loop started at line # 1
```

b) Explanation

Statt 1. Each iteration of this loop selects each feature (f) from a given feature model (FM)

Statt 2. We only set the attributes for those features which are neither selected nor rejected. The condition in statement 2 checks whether the current feature (f) is selected or not.

Statt 3. Assigns null to the feature f that is neither selected nor rejected.

Statt 4. The relevance attribute of f stores the type of relevance feature contains with its parent. For example, a feature may be either of mandatory, optional, Or-group, or Alternative-group.

Statt 5. This statement stores the name of direct parent of the current feature. It will only be null for root feature of FM.

Statt 6. The array (exclude[]) contains the features which are connected to feature (f) by excludes constraint.

Statt 7. The array (requires[]) stores the features which are required by feature (f) through implies constraint.

Statt 8. The array (required_by[]) stores the features that require the selection of feature f by implies constraint.

Statt 9. Check that the current feature has parent feature or not. If the selected feature is not a root feature then total number of alternative set will be counted that falls under the direct parent of current feature.

Statt 10. This condition checks whether a feature (f) is a part of an alternate-group and also selected value is not false, because if feature selection is already false due to any reason then this would not be processed further hence no need to add in alternative list. If condition is true then this feature will be stored in a relative feature list.

Statt 11. To set the alternative list name this variable should have a value because the name of different alternative list will differ by this variable if there are multiple alternative sets under a feature e.g. engine-type_a, Engine-type_b etc.

Statt 12. This function will add the feature in concerned alternative list, if it has an alternative constraint.

Statt 13. If condition started at line # 11 ends.

Statt 14. If condition started at line # 9 ends.

Statt 15. If condition started at line # 2 ends.

(c) Tracing

For the tracing of above mentioned algorithm, we use the feature model shown in Fig. 2. In this example, a feature model of ubuntu software is presented which contains *ubuntu* as root feature. *ubuntu* has *texteditor, bash* and *gui* as mandatory sub-features which exclude each other, while *games* is as optional sub-feature of *ubuntu* that has an implies constraint with *gui*. The feature *texteditor* contains an alternative group of *vi* and *gedit*. Similarly *gui* has an Or-group of *kde* and *gnome*. One more alternative group of *gnuchess* and *glchess* comes under *games* feature.

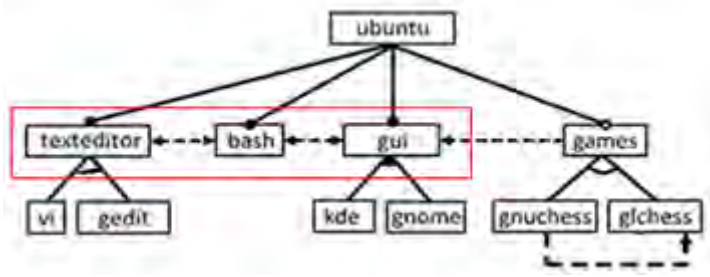

Fig. 2 Void feature model (Felfernig et al., 2013).

Furthermore, the feature *flchess* is connected to *gnuchess* by using implies constraint. We start tracing of our algorithm from the root feature. So, in the first iteration of loop *f* represents *ubuntu*. Also, it is worth mentioning that all features of a given feature model has no attribute value when first time it is inputted to the system. Below is the tracing result of the features in the feature model of our example (Fig. 2) by applying the set attribute example.

Table 1 Attributes values of features after tracing.

Features (f)	**Attributes values**
Ubuntu	selected=null, relevance=null, parent=null, exclude=null, requires=null, required_by=null
Texteditor	selected= null, relevance=mandatory, parent=ubuntu, exclude=[bash], requires=[gnome], required_by=null, ubuntu.total_alternateset=0,

Bash	selected= null, relevance=mandatory, parent=ubuntu, exclude=[texteditor, gui], requires=[kde], required_by=null, ubuntu.total_alternateset=0,
Gui	selected= null, relevance=mandatory, parent=ubuntu, exclude=[bash, games], requires=null, required_by=null, ubuntu.total_alternateset=0,
Games	selected= null, relevance=optional, parent=ubuntu, exclude=gui, requires=null, required_by=null, ubuntu.total_alternateset=0,
Kde	selected=null, relevance=or, parent=gui, exclude=null, requires=null, required_by=[bash], gui.total_alternateset=0,
Gnome	selected= null, relevance =or, parent=gui, exclude=null, requires=null, required_by=[texteditor], gui.total_alternateset=0,
Gnuchess	selected= null, relevance=alternative, parent=games, exclude=null, requires=glchess, required_by=null, games.total_alternateset=1, (added to alternative set) games_a = [gnuchess]
Glchess	selected= null, relevance=alternative, parent=games, exclude=null, requires=null, required_by= gnuchess, games.total_alternateset=1, (added to alternative set) games_a = [gnuchess | glchess]

5.2 Algorithms to find contradiction

These algorithms are used to find contradictory depicted feature model. These contradictions arise due to the wrong application of crosstree constraints. The contradiction causes various errors in feature models. It is important to highlight all contradictions for the discovery of errors. This contradiction finding algorithm consists of two parts: first part is to find contradictory features due the exclude constraint, while the second part is to find contradictory features due to implies constraint.

5.2.1 Exclude contradiction

Below algorithm is the first part in-order to find contradictory features due to the exclude constraint.

a) Algorithm

```
1   foreach  f  in FM
2     if  f.exclude  ≠ null
3       foreach  ex  in  f.excludes
4         if  f.selected  ≠ false  OR ex.selected ≠ false   // to check both or any one is not false
5           if  f.relevance = mandatory AND  ex.relevance≠mandatory
6             ex.selected  ←  false
7             if  ex.relevance=alternative then   ex.Parent_alternateset.remove(ex)
8           End of if condition started at line # 5
9           if  ex.relevance = mandatory AND  f.relevance≠mandatory
10            f.selected  ←  false
11            if  f.relevance=alternative then  f.Parent_alternateset.remove(f)
12          End of if condition started at line # 9
13          if  ex.relevance = mandatory AND  f.relevance = mandatory
14            f.selected ← false
15              ex.selected ← false
16          End of if condition started at line # 13
```

17	*if ex.relevance ≠ mandatory AND f.relevance ≠ mandatory*
18	*if ex.required_by=null OR (ex.required_by≠null AND ex.required_by.relevance≠mandatory)*
19	*ex.selected ← false*
20	*if ex.relevance=alternative then ex.Parent_alternateset.remove(ex)*
21	*End of if condition started at line # 18*
22	*if f.required_by=null OR (f.required_by≠null AND f.required_by.relevance≠ mandatory)*
23	*f.selected ← false*
24	*if f.relevance=alternative then f.Parent_alternateset.remove(ex)*
25	*End of if condition started at line # 22*
26	*End of if condition started at line # 17*
27	*End of if condition started at line # 4*
28	*End of foreach loop started at line # 3*
29	*End of if condition started at line # 2*
30	*End of if condition started at line # 1*

b) Explanation

Statt 1. Each iteration of this loop selects each feature (f) from a given feature model (FM)

Statt 2. This condition will check that (f) has any exclude constraint or not as this function is based on exclude constraint so if exclude constraint is null then will not proceed further

Statt 3. If selected feature has exclude constraint then this loop will select each feature that has exclude constraint with this feature from the exclude list

Statt 4. This condition is to check that selection of any of the feature that excludes each other is false. As the exclude constraint means that both feature cannot appear together in an instance so if selection of any one or both features is false then exclude constraint already satisfied no need to process further. If the selection of both features is true then further checks will be applied

Statt 5. To check that current feature (f) has a mandatory relevance and other feature (ex) that is excluded by (f) has no mandatory relevance

Statt 6. If (f) has a mandatory relevance and (ex) don't have then selection of (ex) is set to false because both exclude each other

Statt 7. If the relevance of (ex) is not mandatory then it'll be checked that it has alternative relevance if yes then (ex) will be removed from concern alternative list because it's selection is set to false

Statt 8. If condition ends, started at line # 5

Statt 9. To check that current feature (f) doesn't have mandatory relevance and other feature (ex) that is excluded by (f) has mandatory relevance

Statt 10. If (f) don't have mandatory relevance and ex has mandatory relevance then selection of (f) is set to false because both exclude each other

Statt 11. If the relevance of (f) is not mandatory then it'll be checked for alternative relevance. If it has alternative relevance then (f) will be removed from concern alternative list because its selection is set to false

Statt 12. If condition ends that started at line # 9

Statt 13. To check that both features (f) and (ex) both have mandatory relevance

Statt 14. Selection of (f) set to false because both are mandatory so can't appear in the same instance due to exclude relevance

Statt 15. Selection (ex) set to false

Statt 16. if condition ends, started at line # 13

Statt 17. Check that both (f) and (ex) don't have mandatory relevance

Statt 18. Check that (ex) is not required by any other feature by implied constraint or if required then the feature that required (ex) is not mandatory

Statt 19. If (ex) has no implies constraint or not implied by any mandatory feature then set selection to false

Statt 20. If (ex) falls under any alternative set then it will be removed from that set because it's selection is set to false

Statt 21. If condition ends that started at line # 18

Statt 22. Check that (f) is not required by any other feature by implied constraint or if required then the feature that required (f) is not mandatory

Statt 23. If (f) has no implies constraint or not implied by any mandatory feature then set selection to false

Statt 24. If (f) falls under any alternative set then it will be removed from that set because it's selection is set to false

Statt 25. If condition ends that started at line # 22

Statt 26. If condition ends that started at line # 17

Statt 27. If condition ends that started at line # 4

Statt 28. foreach loop ends that started at line # 3

Statt 29. If condition ends that started at line # 2

Statt 30. If condition ends that started at line # 1

c) Tracing

Each algorithm will affect the attributes of features in feature model (Fig. 2) as mentioned earlier so this tracing will start from the previous values of attributes (listed in Table 1 after setting attributes). To explain the functionality of "find exclude contradiction" we first present the step-by-trace one feature (as explanation of statements) then the attributes values after tracing.

Table 2 Tracing of algorithm to find exclude contradiction.

Stat #	**Tracing for the feature "texteditor"**
1	Selected feature is texteditor so **f**=texteditor
2	Exclude is not null so this condition is true
3	A feature selected from exclude array so **ex** = bash
4	As selection of both features (f=texteditor) and (ex=bash) not false so this condition is true.
5	Relevance of both (f) and (ex) is mandatory so this condition is false
6	Will not be executed because condition at line # 5 for this statement is false.
7	Will not be executed because condition at line # 5 for this statement is false.
8	If condition end that started at line # 5
9	Relevance of both (f) and (ex) is mandatory so this condition is false
10	Will not be executed because condition at line # 9 for this statement is false.
11	Will not be executed because condition at line # 9 for this statement is false.
12	If condition end that started at line # 9
13	This condition is true because the relevance of both (f) and (ex)
14	Texteditor.selected= false
15	bash.selected= false

16	If condition end that started at line # 9
17	Relevance of both (f) and (ex) is mandatory so this condition is false
18	Will not be executed because condition at line # 17 for this statement is false.
19	Will not be executed because condition at line # 17 for this statement is false.
20	Will not be executed because condition at line # 17 for this statement is false.
21	Will not be executed because condition at line # 17 for this statement is false.
22	Will not be executed because condition at line # 17 for this statement is false.
23	Will not be executed because condition at line # 17 for this statement is false.
24	Will not be executed because condition at line # 17 for this statement is false.
25	If condition end that started at line # 22
26	If condition end that started at line # 17
27	If condition end that started at line # 4
28	Foreach loop end that started at line # 3
29	If condition end that started at line # 2
30	If condition end that started at line # 1

In the Table below, we present the tracing of "find exclude contradiction" algorithm. After tracing of the said algorithm, the attributes of the feature might change that have exclude constraint on each other. Attributes that have been changed after the tracing of this algorithm are underlined.

Table 3 Attributes values after tracing of exclude contradiction algorithm**.**

Features (f)	**Attributes values**
Ubuntu	**selected**= null, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null
Texteditor	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash], **requires**=[gnome], **required_by**=null, **ubuntu.total_alternateset**=0,
Bash	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[texteditor, gui], **requires**=[kde], **required_by**=null, **ubuntu.total_alternateset**=0,
Gui	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash, games], **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Games	**selected**=null, **relevance**=optional, **parent**=ubuntu, **exclude**=gui, **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Kde	**selected**=null, **relevance**=or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[bash], **gui.total_alternateset**=0,
Gnome	**selected**= null, **relevance** =or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[texteditor], **gui.total_alternateset**=0,
Gnuchess	**selected**= null, **relevance=**alternative, **parent**=games, **exclude**=null, **requires**=glchess, **required_by**=null, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess**]**
Glchess	**selected**= null, **relevance**=alternative, **parent**=games, **exclude**=null, **requires**=null,

	required_by= gnuchess, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess | glchess**]**

5.2.2 Alternative contradiction

This algorithm is the second part of finding contradiction. In this section, we will try to find contradiction caused by implies constraint. The inputs of this algorithm are the alternative lists which were created by set attribute algorithm. Each list consists of those features that lie under the same alternative set. Contradiction arises in different situations, but most commonly occurring situation is features falling under similar sets and are connected by implies constraint, as shown in the Fig. 3(a). Other situation is when a mandatory feature requires multiple features and all required features falls under same alternative set, as shown in Fig. 3(b). This contradiction violates the basic constraint that only one feature can be instantiated from a single alternative set.

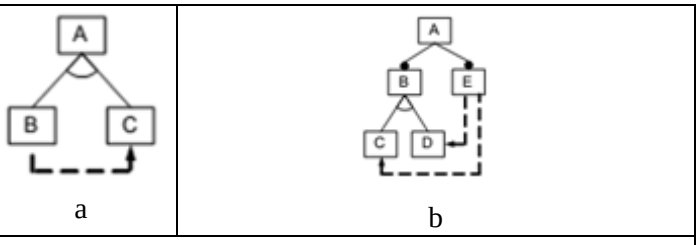

Fig. 3 Contradiction due to alternative relevance (Segura et al, 2010).

a) Algorithm

```
1   foreach alternateset in f.parent_alternatesets
2       total_required ← 0
3       foreach fa in alternateset
4           if fa.requires ≠ null
5               foreach require in fa.requires
6                   if require.parent = fa.parent
7                       fa.selected ← false  // if the required feature for this feature is in the same
    alternate set so the requiring feature is false
8                       alternateset.remove(fa)
9                   End of if condition started at line # 6
10              End of foreach loop started at line # 5
11          End of if condition started at line # 4
12          if fa.required_by ≠ null then total_required ++
13      End of foreach lop started at line # 3
14      if total_required ≥ 2
15          foreach fa in alternateset
16              foreach P_require in fa.required_by
17                  foreach fs in alternateset
18                      if fa ≠ fs
19                          foreach require in fs.required_by
20                              if P_require = require then require.selected=false
```

21	*End of foreach loop started at line # 19*
22	*End of if condition started at line # 18*
23	*End of foreach loop started at line # 17*
24	*End of foreach loop started at line # 16*
25	*End of foreach loop started at line # 15*
26	*End of if condition started at line # 14*
27	*End of foreach loop started at line # 1*

b) Explanation

Statt 1. This loop will select each alternative set from all alternative sets, created by attributes setting algorithm on feature model

Statt 2. Set total required variable to 0. This variable is to store the number of features that are required by a feature by implies constraint

Statt 3. This loop is to get all features one-by-one stored in selected alternative list

Statt 4. This is to check that current feature (fa) from alternative list requires any other feature or not by implies constraint

Statt 5. This loop is to select those features which are required by current feature (fa) by implies constraint

Statt 6. This condition is to check that parent feature of current feature (fa) and the feature which is required by current feature (require) is same. As depicted in Fig. 3(a)

Statt 7. If the required feature of (fa) lies under same alternative set then selection of (fa) will be set false. Because this violates the basic rule of alternative set i.e. only one feature can be instantiated from an alternative set

Statt 8. As (fa) selection is set to false then this feature must be removed from alternative set

Statt 9. if condition ends, started at line # 6

Statt 10. foreach loop ends, that started at line # 5

Statt 11. if condition ends, started at line # 4`

Statt 12. This condition is to check that is there any feature required by other feature using implied constraint if yes then increment the value of total_required variable. The reason for this check and increment is if multiple features from the same alternative set are required then there is a chance of contradiction as in Fig. 3 (b)

Statt 13. foreach loop ends, started at line # 3

Statt 14. Check that total_required value is greater than or equal to 2 or not

Statt 15. This will select each feature from alternative list

Statt 16. Select those features that require the feature (fa) from alternative set

Statt 17. Select feature (fs) one-by-one from the same alternative set

Statt 18. As foreach loops at line # 16 and 18 both are selecting features from the same alternative set so this condition is to check that not the same feature are being compared with each other

Statt 19. Select those feature which require (fs) by implies constraint

Statt 20. If the feature that require (fa) also requiring (fs) then that feature's selection will be set to false because it is violating implying multiple features from a single alternative set

Statt 21. End of foreach loop started at line # 19

Statt 22. End of if condition started at line # 18

Statt 23. End of foreach loop started at line # 17

Statt 24. End of foreach loop started at line # 16
Statt 25. End of foreach loop started at line # 15
Statt 26. End of if condition started at line # 14
Statt 27. End of foreach loop started at line # 1

c) Tracing

This algorithm is to find contradiction due to alternative constraint. So in this tracing we are considering the alternative list that is created during set attributes algorithm. This alternative list is as under:

games_a = [*gnuchess* **|** *glchess***]** this alternative set lies under *games* feature so list name is games_a.

Table 4 Tracing of algorithm to find alternative contradiction.

	Stat #	**Tracing of alternative list games_a**
	1	In this loop games_a alternative set selected
	2	Total_required = 0
First Iteration of foreach loop at line # 3	3	This loop will select first feature from alternative set so **fa=gnuchess**
	4	This condition is **true** as this features requires glchess
	5	As gnuchess requires glches so **require = glchess**
	6	This condition is true because both features (fa) and (require) falls under same parent games
	7	Set the (fa=gnuchess) selection to false due to contradiction
	8	As (fa=gnuchess) selection is set to false so this feature will be romoved from alternative list
	9	End of if condition started at line # 6
	10	End of foreach loop started at line # 5
	11	End of if condition started at line # 4
	12	This condition is false
Second Iteration of foreach loop at line # 3	3	This loop will select second feature from alternative set so **fa=glchess**
	4	This condition is **False** as this features requires no feature
	5~8	These conditions will not execute
	9	End of if condition started at line # 6
	10	End of foreach loop started at line # 5
	11	End of if condition started at line # 4
	12	This condition is true so the value of **total_required=1**
	13	End of foreach loop started at line # 3
	14	This condition is false because **total_required=1**
	15~26	These conditions will not execute
	27	End of first iteration of foreach loop started at line # 1

This algorithm is to find contradiction in alternative sets after tracing. So, mostly features with alternative relevance affected. In our example there are two features that have alternative relevance *gnuchess* and *glchess*

lies under games that contain contradiction because *gnuchess* implies *glchess* so the selection *gnuchess* is set to false. There is no change in the attributes of the remaining three features. After the tracing of this algorithm, values of attributes for the features from our example feature model (Fig. 2) is as under. Attributes that has been changes after the tracing of this algorithm are underlined.

Table 5 Attributes values after the tracing of algorithm "alternative contradiction"

Features (f)	**Attributes values**
Ubuntu	**selected**= null, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null
Texteditor	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash], **requires**=[gnome], **required_by**=null, **ubuntu.total_alternateset**=0,
Bash	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[texteditor, gui], **requires**=[kde], **required_by**=null, **ubuntu.total_alternateset**=0,
Gui	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash, games], **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Games	**selected**=null, **relevance**=optional, **parent**=ubuntu, **exclude**=gui, **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Kde	**selected**=null, **relevance**=or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[bash], **gui.total_alternateset**=0,
Gnome	**selected**= null, **relevance** =or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[texteditor], **gui.total_alternateset**=0,
Gnuchess	**selected**=false, **relevance=**alternative, **parent**=games, **exclude**=null, **requires**=glchess, **required_by**=null, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess**]**
Glchess	**selected**= null, **relevance**=alternative, **parent**=games, **exclude**=null, **requires**=null, **required_by**= gnuchess, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess | glchess**]**

5.3 Algorithm for the selection and rejection of features

This algorithm sets the selection attribute of all features in the given feature model. After tracing of this algorithm, features either be selected or rejected on the basis of constraint applied on the features. The section algorithm (explained below) process all features to find out the selectable features from a given feature model. This will also help to find which feature is not selectable and also help to detect errors in that feature model.

a) Algorithm

```
1  foreach f in FM
2      if f. parent = null AND f.selected ≠ false then f.selected ← true
3      if f.parent.select = true AND f.selected = null AND f. parent ≠ null
4          if f.relevance ≠ alternate
5              if f.requires ≠ null
6                  foreach require in f.requires
```

```
7                       if require.selected ≠ false
8                           f.selected ← true
9                           require.selected ← true
10                      End of if condition started at line # 7
11                      if require.selected = false then f.selected ← false
12                  End of foreach loop started at line # 6
13              else
14                  f.selected ← true
15              End of if-else condition started at line # 5
16          End of if condition started at line # 4
17          if f.relevance = alternative AND f.parent.total_alternatesets ≥ 1
18              select ←false
19              foreach fs in f.parent_alternatesets
20                  if fs.selected = true then select = true
21              End of foreach loop started at line # 19
22              if select = false // select if there is any required is mandatory
23                  foreach fs in f.parent_alternatesets
25                      if fs.required_by ≠ null
26                          foreach require in fs.required_by
27                              if require.selected ≠ false AND if require.relevance=mandatory
28                                  fs.selected ← true
29                                  require.selected ← true
30                                  select ←true
31                                  break // exit from the loop for this alternate set
32                              End of if condition started at line # 27
33                          End of foreach loop started at line # 26
34                      End of if condition started at line # 25
35                  End of foreach loop started at line # 23
36                  if select = false // select if there is no any required is mandatory
37                      foreach fs in f.parent_alternatesets
38                          selected_required_by ←0
39                          if fs.required_by ≠ null
40                              foreach require in fs.required_by
41                                  if require.selected ≠ false
42                                      fs.selected ← true
43                                      require.selected ← true
44                                      select ←true
45                                      break // exit from the loop for this alternate set
46                                  End of if condition started at line # 41
47                              End of foreach loop started at line # 40
48                          End of if condition started at line # 39
49                      End of foreach loop started at line # 37
50                  End of if condition started at line # 36
51                  if select = false
```

```
52                  foreach fs in f.parent_alternatesets
53                      if fs.requires ≠ null
54                          foreach require in fs.requires
55                              if require.selected ≠ false
56                                  fs.selected ← true
57                                  require.selected ← true
58                                  select ← true
59                                  break // exit from the loop for this alternate set
60                              End of if condition started at line # 55
                                if require.selected = false then fs.selected ← false
61                          End of foreach loop started at line # 54
62                      End of if condition started at line # 53
63                  End of foreach loop started at line # 52
64              End of if condition started at line # 51
65              if select = false
66                  foreach fa in alternateset
67                      if fa.selected ≠ false
68                          fa.selected ← true
69                          select ← true
70                          break // exit from the loop for this alternate set
71                      End of if condition started at line # 68
72                  End of foreach loop started at line # 67
73              End of if condition started at line # 66
74              if select = true
75                  foreach fs in f.parent_alternateset
76                      if fs.selected ≠ true then fs.selected ← false
78                  End of foreach loop started at line # 76
79              End of if condition started at line # 75
80          End of if condition started at line # 22
81      End of if condition started at line # 17
82  End of if condition started at line # 3
83  End of foreach loop started at line # 1 // deselect all those which are not selected
84  foreach f in FM
85      if f.selected = null then f.selected = false
86  End of foreach loop started at line # 84
```

b) Explanation

Statt 1. This loop will select features from feature model one-by-one

Statt 2. This condition is to check that (f) has any parent feature or not if it don't have then it is root feature the selection of root feature set to true if the selection of root feature was not set to false due to any constraint

Statt 3. In this condition first to check that that the selection of parent features of (f) is true because a feature can't be instantiated if its parent's selection is false. In second part of this condition it is checked that the selection of (f) is not set to false due to any constraint

Statt 4. To check that (f) does not have alternative relevance

Statt 5. This is to check that does this features has implies constraint for any other feature?

Statt 6. This loop will select each feature that is required by (f) by implies constraint

Statt 7. This is to check that selection of the feature which is required by (f) is not set to false

Statt 8. If implied feature's selection is not set false then the selection of (f) is set to true

Statt 9. Selection of required feature also set to true because without this feature (F) cannot be instantiated

Statt 10. If condition ends, started at line # 7

Statt 11. This is to check that if selection of implied feature is set to false then selection of (f) also set to false because without required feature (f) cannot be instantiated

Statt 12. foreach loop ends, started at line # 6

Statt 13. If the condition of statement # 5 false then coming statements will be executed

Statt 14. Else part of if condition at line # 7

Statt 15. As (f) does not have any constraint or contradiction then (f) set to true

Statt 16. If condition ends that started at line # 5

Statt 17. If condition ends that started at line # 4

Statt 18. This is to check that (f) has alternative relevance and parent feature of (f) have alternative sets

Statt 19. A boolean variable select set to true. This variable will be used to track that any of the feature from an alternative set selected or not because only one feature can be selectable from an alternative set

Statt 20. This loop will select each feature from alternative set as (fs)

Statt 21. To check that if selection of (fs) is true then set the value of select true to keep record of selected feature in alternative set

Statt 22. Foreach loop ends, started at line # 19

Statt 23. To check the value of variable select if true then proceed further

Statt 24. This loop will select each feature from alternative set as (fs)

Statt 25. This to check that (fs) is implied by any other feature or not

Statt 26. This loop will select those features one-by-one which required (fs) by implies constraint

Statt 27. This condition will check that (require) feature's selection is not set to false and its relevance is mandatory

Statt 28. Selection of (fs) set to true because it is required by a mandatory feature (require)

Statt 29. As (require) feature is mandatory so selection of this feature set to true because its required feature is also selected

Statt 30. Value of variable select is set to true because one feature from alternative set has been selected

Statt 31. This break statement is to exit from all loops that are used for this alternative set and will move to statement # 75

Statt 32. If condition ends, started at line # 27

Statt 33. Foreach loop ends, started at line # 26

Statt 34. If condition ends, started at line # 25

Statt 35. Foreach loop ends, started at line # 23

Statt 36. To check the value of select variable that it is true or false

Statt 37. This loop will select each feature from alternative set as (fs) for those feature which require (fs) and don't have mandatory relevance

Statt 38. This to check that (fs) is implied by any other feature or not

Statt 39. This loop will select those features one-by-one which required (fs) by implies constraint (this time loops is to find implied feature by an optional feature)

Statt 40. This condition will check that (require) feature's selection is not set to false

Statt 41. Selection of (fs) set to true because it is required by a feature

Statt 42. Selection of this (require) feature set to true because its required feature is also selected

Statt 43. Value of variable select is set to true because one feature from alternative set has been selected

Statt 44. This break statement is to exit from all loops that are used for this alternative set and will move to statement # 75

Statt 45. If condition ends, started at line # 41

Statt 46. foreach loop ends, started at line # 40

Statt 47. If condition ends, started at line # 39

Statt 48. foreach loop ends, started at line # 37

Statt 49. If condition ends, started at line # 36

Statt 50. To check the value of select variable that it is true or false

Statt 51. This loop will select each feature from alternative set as (fs) for those which are implied by (fs)

Statt 52. This to check that (fs) implies any other feature or not

Statt 53. This loop will select those features one-by-one which are implied by (fs)

Statt 54. This condition will check that (require) feature's selection is not set to false

Statt 55. Selection of (fs) set to true

Statt 56. Selection of this (require) feature set to true

Statt 57. Value of variable select is set to true because one feature from alternative set has been selected

Statt 58. This break statement is to exit from all loops that are used for this alternative set and will move to statement # 75

Statt 59. If condition ends, started at line # 55

Statt 60. This condition will check if implied feature by (fs) is false then (fs) also set to false

Statt 61. foreach loop ends, started at line # 54

Statt 62. If condition ends, started at line # 53

Statt 63. foreach loop ends, started at line # 52

Statt 64. If condition ends, started at line # 51

Statt 65. To check the value of select variable that it is true or false

Statt 66. This loop will select each feature from alternative set as (fa) for those which are neither implied by any feature not (fa) implies any feature

Statt 67. To check that selection of (fa) is not false

Statt 68. Selection of (fa) set to true

Statt 69. Value of variable select is assigned true to indicate that a feature from alternative set has been selected

Statt 70. This break statement is to exit from all loops that are used for this alternative set and will move to statement # 75

Statt 71. If condition ends, started at line # 68

Statt 72. foreach loop ends, started at line # 67

Statt 73. If condition ends, started at line # 66

Statt 74. To check value of select variable true or false

Statt 75. This loop will select each feature from alternative set as (fs)

Statt 76. To check that any of the feature from alternative set is selected or not. If not selected then select a feature

Statt 77. foreach loop ends, started at line # 76

Statt 78. If condition ends, started at line # 75

Statt 79. If condition ends, started at line # 22

Statt 80. If condition ends, started at line # 15

Statt 81. If condition ends, started at line # 03

Statt 82. foreach ends, started at line # 1

Statt 83. This loop will select each feature (f) from feature model. This loop will deselect those features which are not selected due to any reason

Statt 84. This condition will check that if the selection of a feature (f) is neither true nor false then set the selection to false

Statt 85. foreach loop ends, started at line # 84

c) Tracing

This algorithm is to set the selection value true or false. For more explanation of the algorithm by tracing we are using the feature model of our example (Fig. 2). First we are tracing step-by-step of root feature *ubuntu* then of another feature *games*.

Table 6 Tracing of "selection" algorithm for feature "ubuntu"

Stat #	**Tracing**
1	f = ubuntu
2	(as both conditions are true i.e. (f) neither has parent feature not its selection is false) ubnuntu.selected =true
3	This condition is false so reset of the statements will not execute

After tracing of "selection" algorithm, value of "selected" attribute for each feature in a feature model will either be true or false based on the constraints, relevance and parent feature's attributes value. After tracing of this algorithm values of attributes for the features from our example feature model (Fig. 2) is as under. Selected attributes that are changed after the tracing of this algorithm are underlined.

Table 7 Attributes values after tracing of "selection" algorithm.

Features (f)	**Attributes values**
Ubuntu	**selected**=true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null
Texteditor	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash], **requires**=[gnome], **required_by**=null, **ubuntu.total_alternateset**=0,
Bash	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[texteditor, gui], **requires**=[kde], **required_by**=null, **ubuntu.total_alternateset**=0,
Gui	**selected**=null, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash, games],

	requires=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Games	**selected**=null, **relevance**=optional, **parent**=ubuntu, **exclude**=gui, **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Kde	**selected**=false, **relevance**=or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[bash], **gui.total_alternateset**=0,
Gnome	**selected**=false, **relevance** =or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=[texteditor], **gui.total_alternateset**=0,
Gnuchess	**selected**=false, **relevance=**alternative, **parent**=games, **exclude**=null, **requires**=glchess, **required_by**=null, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess**]**
Glchess	**selected**=false, **relevance**=alternative, **parent**=games, **exclude**=null, **requires**=null, **required_by**= gnuchess, **games.total_alternateset**=1, (added to alternative set) **games_a = [**gnuchess | glchess**]**

5.4 Finding maturity level of a feature model

In this section, we define the mechanism for maturity level detection (error detection on each level) in detail. This is algorithmic based mechanism. Errors defined on each level are to be detected as per definition. For each error different algorithm defined and these algorithms will indicated the level if error exist.

5.4.1 Instanceable (Level-0)

A feature model lies on this level if it is a void feature model. To find that feature model lies on this level or not "void feature model "algorithm will be used.

Void feature model

After setting all attributes specially selection attribute it is easy to detect errors. If selection of all features is set false then it will be a void feature model.

a) Algorithm

```
1  Select ← 0
2  foreach f in FM
3    if f.selected = true then select ++
4  End of foreach loop started at line # 2
5  if select < 2 then "The given feature model is at Level 0 (Instanceable level)"
```

b) Explanation

Statt 1. Initializing the select attribute by zero. This variable is used to count the number of selected features

Statt 2. This loop is to select each feature from feature model one-by-one

Statt 3. This condition will check that if (f) is selected then increment the value of select variable used to count selected features

Statt 4. Foreach loop ends, started at line # 2

Statt 5. This condition will check that "are selected features less than 2" because in void feature model selected feature is root feature and some time root feature also not selected. So, in either case number of selected features in a feature model is zero or one (less than 2). If number selected feature are less than 2 in any of the feature model is said to be void and no instance can be generated

c) Tracing

Table 8 Attributes values of feature model shown in Fig. 2.

Features	Attributes values
Ubuntu	**selected**= true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null,
Texteditor	**selected**= false, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash], **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Bash	**selected**= false, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[texteditor, gui], **requires**=null, **required_by**=null, **ubuntu.total_alternateset**=0,
Gui	**selected**= false, **relevance**=mandatory, **parent**=ubuntu, **exclude**=[bash], **requires**=null, **required_by**=games, **ubuntu.total_alternateset**=0,
Games	**selected**= false, **relevance**=optional, **parent**=ubuntu, **exclude**=null, **requires**=gui, **required_by**=null, **ubuntu.total_alternateset**=0,
Vi	**selected**= false, **relevance**=alternative, **parent**=texteditor, **exclude**=null, **requires**=null, **required_by**=null, **texteditor.total_alternateset**=0, (added to alternative list) **texteditor_a = [** vi **]**
Gedit	**selected**= false, **relevance**=alternative, **parent**=texteditor, **exclude**=null, **requires**=null, **required_by**=null, **texteditor.total_alternateset**=0, (added to alternative list) **texteditor_a = [** vi | gedit **]**
Kde	**selected**= false, **relevance** =or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=null, **gui.total_alternateset**=0,
Gnome	**selected**= false, **relevance** =or, **parent**=gui, **exclude**=null, **requires**=null, **required_by**=null, **gui.total_alternateset**=0,
Gnuchess	**selected**= false, **relevance=**alternative, **parent**=games, **exclude**=null, **requires**=glchess, **required_by**=null, **games.total_alternateset**=1, (added to alternative list) **games_a = [**gnuchess**]**
Glchess	**selected**= false, **relevance**=alternative, **parent**=games, **exclude**=null, **requires**=null, **required_by**=gnuchess, **games.total_alternateset**=1, (added to alternative list) **games_a = [**gnuchess | glchess**]**

After the tracing of selection algorithm all feature's selected attributes in our example feature model (Fig. 2) are false due to contradictions and constraints except root feature so it is said to be a void feature model as a result this feature model is said to be at instantiated level.

5.4.2 Acceptable (Level-1)

A feature model will be at this level if the instances (products) generated are not valid. An invalid product is that which is missing any of the mandatory features because mandatory feature are necessary part of a valid product.

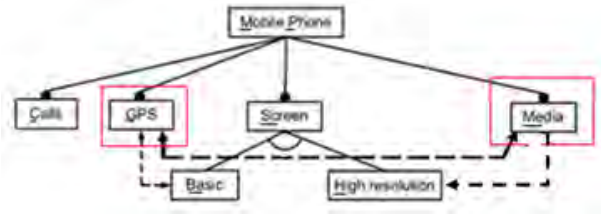

Fig. 4 Feature model with invalid product (Benavides, 2010).

Invalid product

A product missing a mandatory feature of a feature diagram is considered to be an invalid product of that feature diagram (Trinidad, 2008). In the following algorithm, we check invalid products of a given feature diagram.

a) Algorithm

1	*mandatory* $\leftarrow 0$
2	*foreach f in FM*
3	*if f.relevance=mandatory AND f.selected* $\neq$ *true then mandatory ++*
4	*End of foreach loop started at line # 2*
5	*if mandatory* ≥ 1 *then "This Feature Model lies on Acceptable level"*

b) Explanation

Statt 1. Assigning the value to mandatory variable zero. This variable is used to count that how many feature selected

Statt 2. This loop is to select each feature from feature model one-by-one

Statt 3. This condition will check if relevance of (f) is mandatory and its selection is false then increase the value of mandatory variable

Statt 4. Foreach loop ends, started at line # 2

Statt 5. If the value of mandatory variable is greater than or equal to one (one or more mandatory features are not selected) then this feature model will generate invalid product. Hence it is on Acceptable level (Level-1)

c) Tracing

Example of feature model mention in Fig. 4 will be used for the tracing of algorithm to find invalid products. This feature model contains a contradiction in the form of exclude constraint on two mandatory features GPS and Media.

After the tracing of basic attribute setting algorithm, contradiction finding algorithm and selection algorithm the attribute values of features in feature model (Fig. 4) are as under.

Table 9 Attribute values of feature model.

Features	**Attribute values**
Mobile Phone	**selected**= true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null,

Calls	**selected**= true, **relevance**=mandatory, **parent**=Mobile Phone, **exclude**=null, **requires**=null, **required_by**=null,
GPS	**selected**= false, **relevance**=mandatory, **parent**=Mobile Phone, **exclude**=[Media, Basic], **requires**=null, **required_by**=null,
Screen	**selected**= true, **relevance**=mandatory, **parent**=Mobile Phone, **exclude**=null, **requires**=null, **required_by**=null,
Media	**selected**= false, **relevance**=mandatory, **parent**=Mobile Phone, **exclude**=[GPS, High Resolution], **requires**=null, **required_by**=null,
Basic	**selected**= false, **relevance**=alternative, **parent**=Screen, **exclude**=[GPS], **requires**=null, **required_by**=null,
High Resolution	**selected**=true, **relevance**=alternative, **parent**=Screen, **exclude**=Media, **requires**=null, **required_by**=null,

In this feature model both features GPS and Media have mandatory relevance, so these should be the part of every instance. Due to the presence of excludes constraint between GPS and Media, they both cannot be chosen in a single instance and as a result the products generated from this feature model will be invalid. As this feature model contains invalid product error so it lies on acceptable level.

5.4.3 Managed (Level-2)

A given feature model will be on this level if it contains either of dead feature, conditionally dead feature and false variable feature. We propose separate algorithms for the detection of each error.

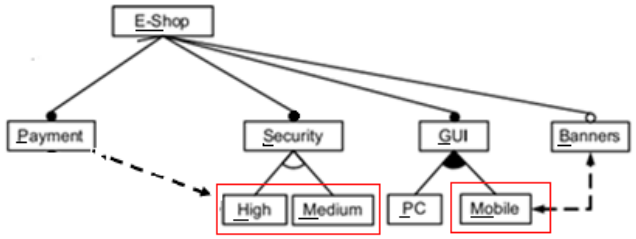

Fig. 5 Feature model with dead, false variable features and conditionally dead feature (Segura et al., 2010)

5.4.3.1 Dead feature

The following algorithm, checks the existence of dead features in a given feature model. If a feature model contains dead feature it will be at level-2.

a) Algorithm

```
1  Dead ←0
2  foreach f in FM
3    if f.selected=false AND (f.exclude.relevance= mandatory OR f.requires.selected=false) then dead++
4      if f.selected=false AND f.relevance =alternative AND f.exclude = null
5        required ←false
6        foreach fa in f.alternateset
7          if fa.selected =true AND fa.required.relevance =mandatory
```

8	*required ← true*
9	*break*
10	*End of if condition started at line # 7*
11	*End of foreach loop started at line # 6*
12	*if required =true*
13	*foreach fa in f.alternateset*
14	*if fa.selected =false then dead ++*
15	*End of foreach loop started at line # 13*
16	*End of if condition started at line # 12*
17	*End of if condition that was started at line # 4*
18	*End of foreach loop that was started at line # 2*
19	*if dead ≥ 1 then "This Feature Model is at Managed Level"*

b) Explanation

Statt 1. Initialize the variable dead to 0. This variable will be used count the dead features found in a feature model

Statt 2. This loop will select each feature (f) from feature model

Statt 3. This condition is to check that whether the selection of (f) is false due to the exclude constraint with mandatory feature or due to implies constraint (selection of implied feature was also false) then this is a dead feature and increment the value of variable dead

Statt 4. This condition is to check that the selection of (f) is false, its relevance is alternative and not excluded by any feature

Statt 5. Initialize the variable required with false. This variable will be used to track that a selected feature from this alternative set is due to implies constraint by a mandatory feature or not.

Statt 6. This loop will select each feature from alternative set

Statt 7. This condition is to check that (fa) is selected due to implies constraint by a mandatory feature

Statt 8. Set required to true if (fa) is selected due to implies constraint by a mandatory feature.

Statt 9. This is to exit from the loop for alternative set when selected feature from alternative set found

Statt 10. If conditions ends, started al line # 7

Statt 11. Foreach loop ends, started at line # 6

Statt 12. This condition is check value of selected variable which was assigned to true when selected feature found in alternative set.

Statt 13. To select all features (fa) from alternative set

Statt 14. If the feature (fa) from alternative set is not selected then it'll be marked dead because of implies constraint from mandatory feature and value of variable dead incremented

Statt 15. Foreach loop ends, started at line # 13

Statt 16. If condition ends, started at line # 12

Statt 17. If condition ends, started at line # 4

Statt 18. Foreach loop ends, started at line # 2

Statt 19. This condition is to check the value of variable dead. If it is greater than or equal to 1 this means this feature model contain dead features so it on level-managed.

c) Tracing

As shown in Fig. 5, there is excludes constraint between *Payment* and *High* features, i.e., they both must not be chosen in one instance. *Payment* feature is mandatory at global level of the diagram—it must be the part of each instance, as result feature *High* will not be select because it has variable relevance and become dead feature.

After the tracing of basic attribute setting algorithm, contradiction finding algorithm and selection algorithm the attribute values of features in feature model (Fig. 5) are as under.

Table 10 Features attribute values of feature model shown in Fig.5.

Features	**Attribute values**
E-Shop	**selected**=true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null,
Payment	**selected**=true, **relevance**=mandatory, **parent**=E-Shop, **exclude**=High, **requires**=null, **required_by**=null,
Security	**selected**=true, **relevance**=mandatory, **parent**=E-Shop, **exclude**=null, **requires**=null, **required_by**=null,
GUI	**selected**=true, **relevance**=mandatory, **parent**=E-Shop, **exclude**=null, **requires**=null, **required_by**=null,
Banners	**selected**= true, **relevance**=optional, **parent**=E-Shop, **exclude**=Mobile, **requires**=null, **required_by**=null,
High	**selected**=false, **relevance**=Alternative, **parent**=Security, **exclude**=Payment, **requires**=null, **required_by**=null,
Medium	**selected**=true, **relevance**=Alternative, **parent**=Security, **exclude**=null, **requires**=null, **required_by**=null,
PC	**selected**= true, **relevance**=or, **parent**=GUI, **exclude**=null, **requires**=null, **required_by**=null,
Mobile	**selected**=false, **relevance**=or, **parent**=GUI, **exclude**=Banners, **requires**=null, **required_by**=null,

Table 11 Tracing of algorithm to find dead feature for the feature “high”

Stat #	**Tracing**
1	Dead =0
2	(f) = Medium
3	This condition is false because exclude is null
4	This condition is true because exclude is null this feature has alternative relevance
5	Set required = false
6	This loop will select each feature from the alternative set to which feature “Medium” belongs so fa=High
7	This condition is true because fa=High is selected and it has implies constraint by a mandatory feature
8	Set required=true
9	This break will end the loop started at line # 6
10	If of line # 7 conditions ends

11	Loop of line # 6 ends
12	Condition is true because required=true
13	This loop will select each feature from the alternative set to which feature “Medium” belongs so fa=High
14	This condition will check if the selection of a feature from alternative set is flase then increment the value of variable dead. In this case dead=1
15	End of foreach loop that was started at line # 13
16	End of if that was started at line # 12
17	End of if that was started at line # 4
18	End of foreach loop that was started at line # 2
19	This condition id true because value of variable dead =1 so “This Feature Model is at Managed Level”

5.4.3.2 Conditionally dead feature

This algorithm is to detect this error. If a given feature model contains conditionally dead features it will be on level-2 (managed).

a) Algorithm

```
1   cdead ← 0
2   foreach f in FM
3     if f.selected=false AND (f.exclude.relevance≠mandatory OR f.requires.selected= false) then cdead++
4       if f.selected =false AND f.relevance = alternative AND f.exclude = null
5         required ← false
6         foreach fa in f.alternateset
7           if fa.selected=true AND fa.required.relevance≠mandatory
8             required ← true
9             break
10          End of if condition that was started at line # 7
11        End of foreach loop that was started at line # 6
12        if required =true
13          foreach fa in f.alternateset
14            if fa.selected =false then cdead ++
15          End of foreach loop that was started at line # 13
16        End of if condition that was started at line # 12
17      End of if condition that was started at line # 4
18  End of foreach loop that was started at line # 2
19  if cdead ≥ 1 then “This Feature Model is at Managed Level”
```

b) Explanation

Statt 1. Initialize the variable cdead to 0. This variable will be used count the dead features found in a feature model

Statt 2. This loop will select each feature (f) from feature model

Statt 3. This condition is to check that whether the selection of (f) is false due to the exclude constraint with non-mandatory feature or due to implies constraint (selection of implied feature was also false) then this is a dead feature and increment the value of variable cdead.

Statt 4. This condition is to check that the selection of (f) is false, its relevance is alternative and not excluded by any feature

Statt 5. Initialize the variable required with false. This variable will be used to track that a selected feature from this alternative set is due to implies constraint by a mandatory feature or not

Statt 6. This loop will select each feature from alternative set

Statt 7. This condition is to check that (fa) is selected due to implies constraint by a non-mandatory feature

Statt 8. Set required to true if (fa) is selected due to implies constraint by a mandatory feature

Statt 9. This is to exit from the loop for alternative set when selected feature from alternative set found

Statt 10. If conditions ends, started al line # 7

Statt 11. Foreach loop ends, started at line # 6

Statt 12. This condition is check value of selected variable which was assigned to true when selected feature found in alternative set

Statt 13. Select all features (fa) from alternative set

Statt 14. If the feature (fa) from alternative set is not selected then it'll be marked dead because of implies constraint from non-mandatory feature and value of variable dead incremented

Statt 15. Foreach loop ends, started at line # 13

Statt 16. If condition ends, started at line # 12

Statt 17. If condition ends, started at line # 4

Statt 18. Foreach loop ends, started at line # 2

Statt 19. This condition is to check the value of variable cdead. If it is greater than or equal to 1 this means this feature model contain conditionally dead features so it on level-managed

c) Tracing

For the tracing of conditionally dead feature algorithm we are using the feature model depicted in Fig. 13 and its features attribute's values mentioned in Table 10.

Table 12 Tracing of algorithm to find "conditionally dead feature"

Stat #	**Tracing on "Mobile" feature**
1	cdead=0
2	f = Mobile
3	This condition is true because selection of (f) is false due to exclude constraint and the relevance of that feature which excludes (f) is not mandatory. Selection of "Banner" feature made it dead. So value of cdead incremented to 1 **(cdead=1)**
4	This condition is false because exclude is not null
5 to 18	These statements will not be executed because the condition at line # 2 is false and all these statements grouped under this condition.
19	This condition is true because the value of cdead=1 because of condition at line # 3 so this feature model contains Conditionally dead feature as a result lies on **Managed level**

5.4.3.3 False variable features

The following algorithm finds the false variable features in a given feature model. If a feature model contains false variable feature then it'll be at managed level.

a) Algorithm

```
1   fvariable ←0
2   foreach f in FM
3       if f.selected=true AND f.relevance≠mandatory AND f.required_by.relevance=mandatory then fvariable++
4       if f.selected=true AND f.relevance=alternate AND f.alternateset.coutfeatures()=2 AND f.parent.relevance =
    mandatory
5           foreach fa in f.alternateset
6               if fa.selected = false AND fa.exclude≠ null AND fa.exclude.relevance= mandatory
7                   fvariable ++
8                   break
9               end of if condition started at line # 6
10          end of foreach loop started at line # 5
11      end of if condition started at line # 4
12  End of foreach loop started at line # 2
13  if fvariable ≥1 then "This Feature Model is at Managed Level"
```

b) Explanation

Statt 1. Initialized a variable fvariable to zero. This will be used to count the false variable features in feature model if exist

Statt 2. This loop will select features (f) from a feature model

Statt 3. This condition is to check that a non-mandatory feature selected due to implies constraint by mandatory feature. So, it is a false variable feature and the value of fvariable incremented

Statt 4. This condition will check that selected non-mandatory feature (f) has alternative relevance with its mandatory parent feature and this alternative set has only two features. This check is to detect exclude constraint implied by a mandatory feature on a feature having alternative relevance. If one feature from two alternative features excluded the defiantly other feature will be selected

Statt 5. This loop will select features (fa) from alternative set

Statt 6. This condition is to check that feature (fa) is not selected due to exclude constraint by a mandatory feature as a result other feature will be selected

Statt 7. One feature is selected automatically in this alternative set so the value of fvariable incremented

Statt 8. This to stop the loop started at line # 5

Statt 9. If condition ends, started at line # 6

Statt 10. Foreach loop end, started at line # 4

Statt 11. If condition ends, started at line # 2

Statt 12. Foreach loop ends, started at line # 5

Statt 13. This condition will check if the value of fvariable is greater than or equal to one then this feature model contains false variable feature model hence it is on level-managed

c) Tracing

This algorithm will be applied on feature model depicted in Fig. 5 to find false variable features. In this feature model, *Medium* is a false variable feature. It'll be the part of all instances generated from this feature model. After tracing of the basic attribute setting algorithm, contradiction finding algorithm and selection algorithm the attribute values of features in feature model (Fig. 5) are listed in Table 11.

Table 13 Tracing of algorithm to find "false variable features" for the feature "medium"

Stat #	**Tracing on the feature "High"**
1	fvariable=0
2	f=High
3	This condition if true because this feature (f) is implied by a mandatory feature so fvariable=1
4	This condition is true because (f) has alternative relevance, this alternative set has two features and parent feature of (f) has mandatory relevance
5	This loop will select feature (fa = *High*) from alternative set
6	This condition is false because (fa= *High*) is selected and exclude is null
7	Will not be executed because condition is false.
8	Exit from the foreach loop started at line # 5 and move to statement # 12
13	This condition is true because value of fvariable=1. It means this feature model contain false variable feature so this feature model is at **Level Managed**

5.4.4 Consistent (Level-3)

Errors due to the redundancy in feature model are placed in this level and the following algorithms use to detect these errors.

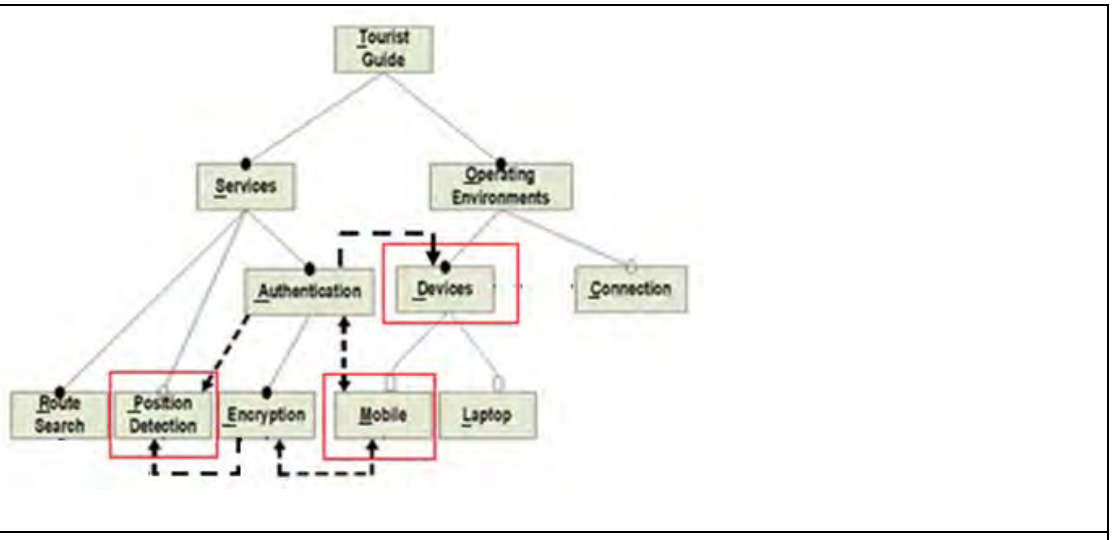

Fig. 6 Feature model with multiple exclusions, multiple implications & implied mandatory (Zhang and Lin, 2011).

5.4.4.1 Multiple exclusions

This algorithm is to find the redundancy caused by multiple exclusions especially by mandatory features.

a) Algorithm

```
1   excluded_feature ← 0
2   foreach f in FM
3       exclude ← 0
4       if f.selected = false AND f.exclude ≠ null
5           foreach ex in f.exclude
6               if ex.relevance = mandatory then exclude ++
7           End of foreach loop started at line # 5
8       End of if condition started at line # 4
9       if exclude ≥ 2 then excluded_feature ++
10  End of forach loop started at line # 2
11  if excluded_feature ≥ 1 then "This Feature Model contains Multiple Excluded Features so it is on Consistent Level "
```

b) Explanation

Statt 1. Excluded_feature initialized to count multiply excluded features in a feature model

Statt 2. This loop will select feature from feature model

Statt 3. Variable exclude initialize to count that how many mandatory features exclude (f)

Statt 4. This is to check that (f) is not selected and excluded by any feature

Statt 5. This loop will select each feature that exclude (f)

Statt 6. This to check that feature that exclude (f) has mandatory relevance then increment the value of exclude variable

Statt 7. Foreach loop ends, started at line # 5

Statt 8. If condition ends, started at line # 4

Statt 9. This is to check that if (f) is excluded by two or more that two mandatory features then it has multiple exclusions. So, the value of excluded_features incremented

Statt 10. Foreach loop end that was started at line # 2

Statt 11. This to check that if there any multiple excluded features then the feature model is at level consistent

c) Tracing

The feature model shown in Fig. 6 is redundantly modelled. First redundancy in this feature model is an optional feature *Position Detection* is implied by two mandatory features *Route Search* and *Authentication*, which is normally called multiple implications (Maßen and Horst, 2004). Secondly, *Mobile* feature is excluded by two mandatory features *Encryption* and *Authentication*, which is normally called multiple exclusions (Maßen and Horst, 2004). Thirdly, feature *Authentication* implies a mandatory feature *Devices*, which is called implied mandatory feature (Maßen and Horst, 2004).

All of the discussed redundancies are detected by the help of different algorithms. Here, in this tracing of algorithm we are going to find multiple exclusion redundancy depicted on feature *Mobile*.

Feature	**Attributes Value**
Tourist Guide	**selected**= true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null,
Services	**selected**= true, **relevance**=mandatory, **parent**=Tourist Guide, **exclude**=null, **requires**=null, **required_by**=null,

Operating Environment	**selected**= true, **relevance**=mandatory, **parent**=Tourist Guide, **exclude**=null, **requires**=null, **required_by**=null,
Route Search	**selected**= true, **relevance**=mandatory, **parent**=Services, **exclude**=null, **requires**=null, **required_by**=null,
Position Detection	**selected**= true, **relevance**=optional, **parent**=Services, **exclude**=null, **requires**=null, **required_by**=[Authentication | Encryption],
Authentication	**selected**= true, **relevance**=mandatory, **parent**=Services, **exclude**=[Mobile], **requires**=[Position detection | Devices], **required_by**=null,
Devices	**selected**= true, **relevance**=mandatory, **parent**=Operation Environment, **exclude**=null, **requires**=null, **required_by**=[Authentication],
Connection	**selected**= true, **relevance**=optional, **parent**=Operation Environment, **exclude**=null, **requires**=null, **required_by**=null,
Encryption	**selected**=true, **relevance**=mandatory, **parent**=Authentication, **exclude**=[Mobile], **requires**=[Position Detection], **required_by**=null,
Mobile	**selected**= False, **relevance**=Alternative, **parent**=Devices, **exclude**=[Authentication | Encryption], **requires**=null, **required_by**=null,

Table 15 Tracing of algorithm to find "multiple exclusions" redundancy.

Stat #	**Tracing on the feature "Mobile"**
1	excluded_feature=0
2	f= Mobile
3	exclude=0
4	This condition is true because (f) is not selected and exclude is not null
5 1st Iteration	ex = Authentication
6 1st Iteration	This condition is true because the relevance of (ex=Encryption) is mandatory so the value of **exclude=1**
5 2nd Iteration	ex= Encryption
6 2nd Iteration	This condition is true because the relevance of (ex=Encryption) is mandatory so the value of **exclude=2**
7	foreach loop ends, started at line # 5
8	If condition ends, started at line # 4
9	This condition is true because the value of exclude=2 so the value of **excluded_features=1**
10	For each loop end that was started at line # 2
11	This condition is true because value of excluded_feature =1 which prove that this feature model contains multiple exclusion redundancy so this feature model lies on level Consistent

5.4.4.2 Multiple implications

This algorithm is to detect the redundancy caused by the multiples implication on variable features.

a) Algorithm

```
1  multi_implies ← 0
2  foreach  f  in  FM
3    mandatory_implies ← 0
4    if f.relevance≠mandatory  AND  f.required_by≠ null
5      foreach  require in  f.required_by
6        if require.relevance=mandatory then mandatory_implies ++
7      End of foreach loop started at line # 5
8    End of if condition started at line # 4
9    if  mandatory_implies ≥ 2  then multi_implies ++
10 End of foreach loop started at line # 2
11 if  multi_inmplies ≥ 1 then  "This Feature Model multiple implications  so it is on Consistent  Level "
```

b) Explanation

Statt 1. Multi_implies initialized to count multiply implied features in a feature model

Statt 2. This loop will select feature from feature model

Statt 3. Variable mandatory_implies initialize to count that how many mandatory features implies (f)

Statt 4. This is to check that (f) is selected variable feature and implied by any feature

Statt 5. This loop will select each feature that implies (f)

Statt 6. This to check that feature that implies (f) has mandatory relevance then increment the value of mandatory_implies variable

Statt 7. Foreach loop ends, started at line # 5

Statt 8. If condition ends, started at line # 4

Statt 9. This is to check that if (f) is implied by two or more that two mandatory features then it is implied by multiple features. So, the value of multi_implies incremented

Statt 10. Foreach loop ends, started at line # 2

Statt 11. This to check that if there any multiple implied feature then the feature model is at level consistent

c) Tracing

This algorithm to find multiple implications applied on feature model in Fig. 6. The attributes values of features for the feature model used as example for this algorithm are in Table 14.

Table 16 Tracing of algorithm to find "multiple implication"

Stat #	**Tracing on the feature "Position Detection"**
1	multi_implies=0
2	f = Position Detection
3	mandatory_implies=0
4	This condition is true because (f) is selected and implied by multiple mandatory features
5 1st Iteration	require=Authentication

6 1st Iteration	This condition is true because (require=Authentication) is a mandatory feature so **mandatory_implies=1**
5 2nd Iteration	require=Encryption
6 2nd Iteration	This condition is true because (require=Encryption) is a mandatory feature so **mandatory_implies=2**
7	foreach loop ends, started at line # 5
8	If condition ends, started at line # 4
9	This condition is true because the value of mandatory_implies=2 so **multi_implies=1**
10	Foreach loop end that was started at line # 2
11	This condition is true because value of multi_implies=1 which proves that this feature model contains the multi implication redundancy so it lies on the level **Consistency**

5.4.4.3 Implied mandatory features

This algorithm is find redundancy due to implied constraint on mandatory feature.

a) Algorithm

```
1  Implied_mandatory ← 0
2  foreach f in FM
3      if f.relevance=mandatory AND f.required_by≠null then implied_mandatory++
4  End of foreach loop started at line # 2
5  if mandatory_implied ≥ 1 then "This Feature Model is on Optimized Level "
```

b) Explanation

Statt 1. Implied_mandatory variable initialized to count implied mandatory features in a feature model

Statt 2. This loop will select feature from feature model

Statt 3. This is to check that (f) is selected mandatory feature and implied by any feature then increment the value of implied_mandatory

Statt 4. Foreach loop ends, started at line # 2

Statt 5. This to check that if there any implied mandatory feature then the feature model is at level Optimized

c) Tracing

This algorithm is to find implied mandatory feature redundancy from feature model in Fig. 6. The attributes values of this feature model are in Table 14 and the Table 17 contains the tracing of the algorithm.

Table 17 Tracing of algorithm to find "implied mandatory feature"

Stat #	**Tracing on the feature "devices"**
1	implied_mandatory=0
2	f= Devices
3	This condition is true because (f=Devices) is a mandatory feature and it is implied by a feature (required_by=Connection) so **implied_mandatory=1**
4	Foreach loop ends, started at line # 2
5	This condition is true because the value of implied_mandatory =1 which shows that this feature

	model contains redundancy of implied mandatory feature this feature model is

5.4.4.4 Duplicate features

This Algorithm is to find multiple features with same names in a feature model.

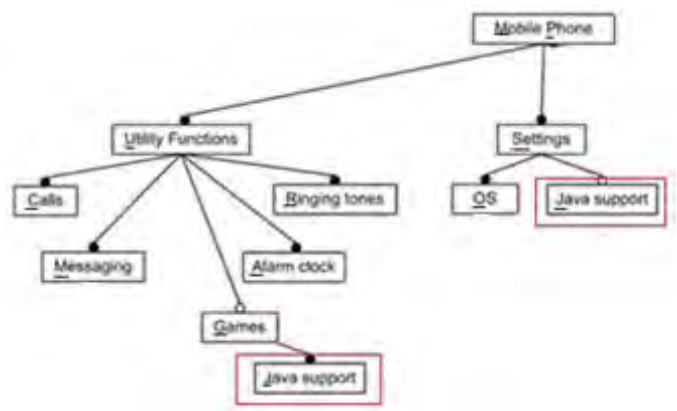

Fig. 7 Feature model with duplicate features (Segura, 2011).

a) Algorithm

```
1  duplicatefeature ←0
2  foreach  f  in  FM
3     dfeature ← 0
4     foreach  fa  in  FM
5        if  f = fa  then  dfeature ++
6     End of foreach loop started at line # 4
7     if  dfeature ≥ 2  then duplicatefeature ++
8  End of foreach loop started at line # 2
9  if  duplicatefeature  ≥ 1 then "This Feature Model contains Duplicate  Features so it is on Consistent
   Level"
```

b) Explanation

Statt 1. Duplicate feature variable initialized to count duplicate feature

Statt 2. This loop will select feature (f) from feature model

Statt 3. dfeature variable initialized to count that (f) has a duplicate feature

Statt 4. This loop will select feature (fa) from feature model. This loop will compare the features.

Statt 5. This condition is to check that if both feature are equal then increment the value of dfeature

Statt 6. Forech loop ends, started at line # 4

Statt 7. This condition is to check that the value dfeature is greater than or equal to 2 increment the value of duplicate feature variable because once the feature (f) will be compared with itself and rest of the time with other so two or greater than two mean this feature has duplicate feature in this feature model

Statt 8. Foreach loop end that was started at line # 2

Statt 9. If a feature model contains duplicate features then it'll be at level consistent

c) Tracing

For the tracing of algorithm to find duplicate features following feature model will be used as an example (Fig. 7). This feature model contains two features with the same name *Java Support*. After the tracing of basic attribute setting algorithm, contradiction finding algorithm and selection algorithm the attribute values of features in feature model (Fig. 6) are as under.

Table 18 Attributes values of features from the feature model in Fig. 7.

Features	**Attributes values of feature model**
Mobile Phone	**selected**= true, **relevance**=null, **parent**=null, **exclude**=null, **requires**=null, **required_by**=null,
Utility Functions	**selected**= true, **relevance**=mandatory, **parent**=Mobile Phone, **exclude**=null, **requires**=null, **required_by**=null,
Calls	**selected**= true, **relevance**=mandatory, **parent**=Utility Function, **exclude**=null, **requires**=null, **required_by**=null,
Messaging	**selected**= true, **relevance**=mandatory, **parent**=Utility Function, **exclude**=null, **requires**=null, **required_by**=null,
Games	**selected**= true, **relevance**=mandatory, **parent**=Utility Function, **exclude**=null, **requires**=null, **required_by**=null,
Alarm Clock	**selected**= true, **relevance**=mandatory, **parent**=Utility Function, **exclude**=null, **requires**=null, **required_by**=null,
Ringing Tones	**selected**= true, **relevance**=mandatory, **parent**=Utility Function, **exclude**=null, **requires**=null, **required_by**=null,
OS	**selected**= true, **relevance**=mandatory, **parent**=Setting, **exclude**=null, **requires**=null, **required_by**=null,
Java Support	**selected**= true, **relevance**=mandatory, **parent**=Setting, **exclude**=null, **requires**=null, **required_by**=null,
Java Support	**selected**= true, **relevance**=Mandatory, **parent**=Games , **exclude**=null, **requires**=null, **required_by**=null,

Table 19 Attributes values of features from the feature model shown in Fig. 7.

Stat #	**Tracing on the feature "Java Support"**
1	duplicatefeature =0
2	f = Java Support (Parent=Games)
3	dfeature =0
4	fa = Java Support (Parent=Games)
5	dfeature =1
4	fa= Java Support (Parent=Setting)
5	Dfeature=2
6	Forech loop end that was started at line # 4
7	This condition is true because the value of dfeature =2 so **duplicatefeature=1**

8	Forech loop end that was started at line # 2
9	This condition is true because duplicatefeature=1 which proves that this feature model contains dup;icate feature so this feature model is at level Consistant

6 Conclusion and Future Work

In this paper, we have presented algorithmic based quality detecting technique for feature models. For the detection of each error, a separate detection algorithm is used. These algorithms evaluate the quality of a given feature model by detecting errors mentioned on each level of the maturity model.

The work is in progress in the following directions:

1. Finding the complexity of these algorithms
2. Verifying the provided semantics with the help of standard interpretation techniques of feature models, like propositional logic, constraint programming etc.
3. We are also planning to extend our work to find the quality levels of the extended version of feature models like, service feature diagrams (Naeem and Heckel, 2011; Naeem, 2012) and cardinality based service e feature diagrams (Assad et al., 2014).

References

Ahmed F, Fernando C. 2011. A business maturity model of software product line engineering. Information Systems Frontiers, 13(4): 543-560

Assad GM, Naeem M, Wahab HA. 2015. Towards cardinality-based service feature diagrams, Computational Ecology and Software: 5(1): 69-76

Batory D, Benavides D, Antonio R. 2006. Automated analysis of feature models: challenges ahead. Communications of the ACM, 49(12): 45-47

Batory D. 2005. Feature models, grammars, and propositional formulas. 9th International Conference on Software Product Lines. 1: 7-20

Benavides D, Segura S, Ruiz-Cortés A. 2010. Automated analysis of feature models 20 years later: A literature review. Information Systems, 35(6): 615-636

Benavides D. 2007. On the automated analysis of software product lines using feature models. PhD Dissertation, Universidad de Sevilla, Spain

Böckle G, Van Der Linden F. 2005. Software Product Line Engineering (Klaus Pohl, ed) Vol. 10. Springer, Heidelberg, Germany

Clements P. 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Felfernig A, Benavides D, Galindo J, Reinfrank F. 2013. Towards anomaly explanations in feature models. 15th International Configuration Workshop. 117- 124

Hemakumar, A. 2008. Finding contradictions in feature models. 12th International Conference on Software Product Line Vol. 2. 183-190

Javed M, Naeem M, Wahab HA. 2014. Towards the maturity model for feature oriented domain analysis, Computational Ecology and Software, 4(3): 170-182

Kang, KC, Cohen SG, Hess JA. Novak WE, Peterson AS. 1990. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Pittsburg, SEI, USA

Maßen T, Horst L. 2004. Deficiencies in feature models. In: Workshop on Software Variability Management for Product Derivation - Towards Tool Support, collocated with the 3rd International Software Product Line Conference. 3154: 331-331, Springer Berlin Heidelberg, Germany

Mendonça M. 2009. Efficient reasoning techniques for large scale feature models. PhD Dissertation, University of Waterloo, Canada

Naeem M., Heckel R. 2011. Towards matching of service feature diagrams based on linear logic. 15th International Conference on Software Product Lines Vol. 2.

Naeem M. 2012. ing of service feature diagrams based on linear logic. PhD Dissertation, Department of Computer Science, University of Leicester, UK

Rosso CD. 2006. Experiences of performance tuning software product family architectures using a scenario-driven approach. 10th International Conference on Evaluation and Assessment in Software Engineering. 1-9

Rubin J, Chechik M. 2012. Combining related products into product lines. 15th International Conference on Fundamental Approaches to Software Engineering (Lara J, Zisman A, eds). 285-300, Springer-Verlag, Berlin, Heidelberg, Germany

Segura S, Benavides D, Ruiz-Cortés A. 2010. FaMa Test Suite v1.2. Technical Report ISA-10-TR-0: 1-52. Applied Software Engineering Research Group, University of Seville, Spain

Segura S. 2011. Extended support for the automated treatment of feature models. PhD Dissertation, University of Sevilla, Spain

Thomas E. 2008. SOA: Principles of Service Design (Vol. 1). Prentice Hall, USA

Thörn C. 2007. A quality model for evaluating feature models. The 11th International Software Product Lines Conference Vol. 2. 184-190, Kindai Kagaku Sha Co. Ltd., Tokyo, Japan

Thörn C. 2010. On the quality of feature models. PhD Dissertation, Department of Computer and Information Science, Linköping University, Sweden

Trinidad P, Benavides D, Durán A, Ruiz-Cortés A, Toro M. 2008. Automated error analysis for the agilization of feature modeling. Journal of Systems and Software, 81(6): 883-896

Zhang G, Ye H, Lin Y. 2011. Feature model validation: A constraint propagation-based approach. 10th International Conference on Software Engineering Reaserch and Practice. Las Vegas, USA