Managing Domain Analysis in Software Product Lines with Decision Tables: An Approach for Decision Representation, Anomaly Detection and Resolution

Nicola Boffoli[a], Pasquale Ardimento[b] and Alessandro Nicola Rana
Department of Informatics, University of Bari, Via Orabona 4, Bari, Italy
{*nicola.boffoli, pasquale.ardimento*}*@uniba.it, a.rana15@studenti.uniba.it*

Keywords: Software Product Lines, Product Derivation, Domain Analysis, Decision Tables, Verification and Validation.

Abstract: This paper proposes an approach to managing domain analysis in Software Product Lines (SPLs) using Decision Tables (DTs) that are adapted to the unique characteristics of SPLs. The adapted DTs enable clear and explicit representation of the intricate decisions involved in deriving each software product. Additionally, a method is presented for detecting and resolving anomalies that may disrupt proper product derivation. The effectiveness of the approach is evaluated through a case study, which suggests that it has the potential to significantly reduce development time and costs for SPLs. Future research directions include investigating the integration of SAT solvers or other methods to improve specific cases of scalability and conducting empirical validation to further assess the effectiveness of the proposed approach.

1 INTRODUCTION

Nowadays companies interested in the development of software applications have to be competitive and highly responsive in a way that they can follow the market trends and the customers' needs to realize flexible products, then they are not specialized on the realization of only one product but they are focused in the development of a large set of products. In this sense Software Product Line Engineering (SPLE) (Clements and Northrop, 2001), (Pohl et al., 2005), (Krueger, 2006) is a powerful instrument, this is a paradigm based on the combination of platforms and mass customization. It is useful to decrease the complexity in developing software-intensive systems and software products by starting from existing ones. Good use of Software Product Line (SPL) implies a thorough understanding of customers needs, organizational context and human behaviors. This knowledge must be elicited and documented to be successfully handled in the product derivation process. Therefore, companies have to manage on one hand all the possible organizational and functional features that the target product must satisfy, on the other hand all the actions needed to implement specific functionalities in a flexible manner. This implies the management of different decisions and even a single wrong decision can easily affect the correct tuning of the product. Sometimes the knowledge necessary to derive new products from a SPL is not clearly explained as it could be described in terms of operations to be executed, functions to be performed and data structures to be used but there is not so much emphasis on rules and constraints. This will give too much freedom of choice to developers and it will lead to the realization of products non-compliants with the functional and non-functional requirements. In this sense, modeling explicitly and correctly the necessary knowledge when developing new applications through a SPL is an important challenge.

In this area, numerous advances have been achieved through practices such as the Feature Oriented Domain Analysis (FODA). The main contribution of FODA (Kang et al., 1990) already in the 90s shed light on the most critical tasks in the context of the analysis of feature models (FM), and their possible automation, through which it is possible to model the variability of the products of an SPL and subsequently investigate the presence of any anomalies and errors in their representation.

In the following years, the FMs have been the subject of studies aimed mainly at guaranteeing:

- the *validity* of FM (aka satisfiability) was immediately identified as a basic need (Mannion, 2002),

[a] https://orcid.org/0000-0001-9899-6747
[b] https://orcid.org/0000-0001-6134-2993

Boffoli, N., Ardimento, P. and Rana, A.
Managing Domain Analysis in Software Product Lines with Decision Tables: An Approach for Decision Representation, Anomaly Detection and Resolution.
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 182-192
ISBN: 978-989-758-647-7; ISSN: 2184-4895
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

(Maßen and Lichter, 2003), (Wang et al., 2005), (Czarnecki and Kim, 2005),(Batory et al., 2006), (Benavides et al., 2013), (Drave et al., 2019), (Galindo and Benavides, 2020), (Horcas et al., 2021), (Le et al., 2021);

- the *internal consistency* of FM, often caused by the presence of "dead features" (Trinidad et al., 2006);
- the *simplification* of the FM, which provides for the possibility of normalizing this model (Van Deursen and Klint, 2002), (Zhang et al., 2004), (von der Maßen and Lichter, 2004);.

In the pure domain of Domain Analysis, the research is mature and solutions and practices can be considered reliable and consolidated. However, there is still a decision-making area that requires in-depth analysis and support, sometimes analyzing exclusively the features and their models can be reductive and/or insufficient. Very often it is necessary to have a combined vision between:

- all the *features of the model* that combine to identify the individual products of an SPLs
- all the *mechanisms of variability* (VM) that must be operationally adopted to generate very single product of an SPLs.

This view can lead to the detection of anomalies due to a much wider range of illegal situations: detection not only of "dead features", but also of incompatible variation mechanisms and specially the detection of anomalies due to the combination of some features of the model with particular cases of variation mechanisms. Broadly speaking, this type of investigation can guide to:

- the correction of the starting *feature model*
- the refinement of the applicable *variation mechanisms*
- to the rearrangement of the *relationships* that link the choice of a set of features to a set of variation mechanisms to be performed in order to generate a specific product of an SPLs

For this purpose, the authors propose an approach based on the use of a framework based on the use of decision tables (DTs) (Maes and Van Dijk, 1988) and (Vanthienen et al., 1998).

Decision Tables (DTs) have been proven to be an effective tool in modeling complex decision-making situations. They provide strategic support for knowledge elicitation by addressing the complexity of the domain and assisting decision-makers in considering the values of business conditions that involve multiple sets of rules. Previous research, such as the studies presented in (Boffoli et al., 2013), (Boffoli et al., 2014), and (Ardimento et al., 2016), have shown the potential of using DTs as a knowledge modeling formalism. The authors of these documents have highlighted the usefulness of DTs in supporting the modeling of complex decision situations, despite their original purpose of supporting programming activities. Therefore, DTs can be used as an approach for understanding and modeling business logic.

The main objective of this paper is to show the potential of using the Decision Table (DT) approach in modeling the decisions that support software engineers during the domain analysis and product derivation stages of a Software Product Line (SPL). The authors propose using DTs as a tool to bridge the gap between the Domain Analysis and Product Derivation processes and support each step towards the creation of final products.

By understanding that software engineers have to follow certain rules to develop an application, which dictate how variation mechanisms are activated to implement functionalities of the final product, based on certain conditions (such as product requirements and production constraints), it is possible to model this process through DTs. This allows for a clear connection between the combination of conditions and the activation of variation mechanisms for the creation of a specific software application.

Additionally, by taking advantage of the validation and verification capabilities of DTs, the authors aim to show that it is possible to prevent, detect, and fix anomalies in the modeled knowledge, even within the context of SPLs. This is achieved by exploiting the unique characteristics of DTs and their ability to model knowledge in SPLs context. Anomalies can occur in the combination of feature values, the methods used to activate variation mechanisms, or the connection between them. By detecting and correcting these anomalies, DTs provide software engineers with accurate, consistent, complete, non-redundant, and easily understandable information, which is crucial in the product derivation process.

The structure of the paper is as follows: In Section 2, we will present the background of DTs and introduce the running example. Section 3 will focus on adapting the DT approach to the context of SPLs and providing a guide for modeling the knowledge necessary for the product development process. In Section 4, we will show how the knowledge validation and verification provided by DTs can be applied in the context of SPLs. Section 5 will discuss potential limitations and challenges. Finally, in Section 6, we will draw conclusions and offer recommendations for future work. Throughout the paper, we will use a running example to illustrate the effectiveness of the proposed approach in an ad-hoc SPL scenario.

2 BACKGROUND

he purpose of this section is two-fold. First, we will provide a thorough overview of the foundations of decision tables, including their structure, elements, and principles. This will serve as a foundation for understanding the rest of the paper. In the second part of this section, we will introduce a running example that will be used throughout the remainder of the paper. This example will provide a concrete illustration of how decision tables can be applied in a specific SPL scenario and help to support the concepts discussed in the rest of the paper.

2.1 Foundations of Decision Tables

Decision tables (DTs) are a powerful tool for representing decision-making processes in a clear and organized manner. Using tabular notation, they match the state of a set of conditions with a corresponding set of actions, as outlined in (Maes and Van Dijk, 1988) and (Vanthienen et al., 1998)). Main advantages of DTs approach, including:

- The *formalization* of knowledge and provision of a compact and customizable view.
- The ease of *maintenance* for business rules as they evolve.

A decision table (DT) is composed of four different quadrants:

- Quadrant 1: the *condition subjects*. It lists all the possible factors that may influence the set of actions to be performed.
- Quadrant 2: the *conditional states*. It lists all the variations of the defined condition subjects that have meaning.
- Quadrant 3: the *action subjects*. It lists all the possible actions that could be performed.
- Quadrant 4: the *action values*. It defines the relationship between a specific combination of conditional states and the corresponding actions to be taken.

About the fourth quadrant it is worth noting that relationships can be expressed either through the quadrant or through an equivalent set of business rules. Indeed, it is often used as a way to graphically represent the required set of business rules.

As depicted in Fig.1, a generic DT is illustrated, featuring the four quadrants and a clear representation of how information should be structured using this schema.

Anyway, ensuring the accuracy of DTs is vital to making sound business decisions. Any anomalies

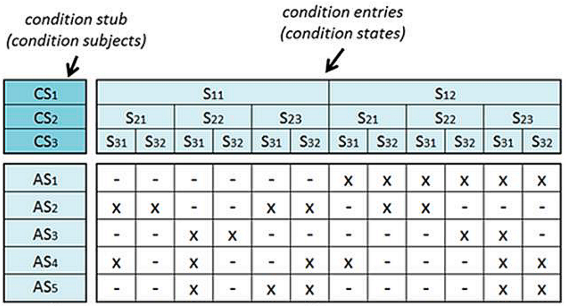

Figure 1: The schema of a decision table.

present in the decision-making process can lead to the construction of DTs that result in poor or ineffective choices. Furthermore, managing a large number of DTs can be a time-consuming and error-prone task, particularly if the information is not well-organized and clear. For the sake of completeness we refer to (Preece and Shinghal, 1994) classification of anomalies that may affect decision tables and corresponding business rules. Below, we provide a summary of this classification.

- *Redundancy* anomalies, while not causing errors, can impede efficiency and complicate maintenance and consistency when revising decision rules. These anomalies can manifest in forms such as subsumption, duplication, unfirable actions, and unnecessary conditions.
- *Inconsistency* anomalies can arise when knowledge is distributed among a plethora of independently designed rules, resulting in issues like conflicting or ambivalent rules.
- *Deficiency* anomalies are a prevalent issue within specific domains, where oversights like unused condition values, unusable actions, and missing rules may occur.

Thankfully DTs have a strenght aptitude to anomalies prevention, detection and fixing. Through the use of the same DTs, the authors in (Boffoli et al., 2014), (Ardimento et al., 2016) show how it is possible to detect and to fix the anomalies by checking rows and columns according to particular roadmap. This activity can be executed automatically or manually. In the first case, it is possible to use a software application that receive as input the DTs to analyze: it will process all the modeled information to check for the existence of anomalies and then, if any issues are found, it will try to fix them by applying the proper criteria for each specific anomaly. The execution of detection and fixing activities on the business knowledge modeled through the DTs could also be carried out manually by experts, who know all the information about the employed actions and the modeled con-

ditions as well as the way to identify the anomalies and how to apply the proper criteria for their fixing. Anyway, as these activities require the cyclical repetition of sequential steps for each rule derived from the DT, the effort may arise when the modeled information starts to grow. For this reason, the manual execution of these tasks is feasible only if the treated case is manageable without the necessity of automation. Therefore, in conclusion, we will show in the rest of the paper that it could be possible to adapt the DTs concept to the SPLs context. In particular, we will employ DTs in SPLs to guarantee that the set of decisions that transforms the software assets in final products is consistent, complete and non-redundant.

2.2 Running Example: Journalling Flash File System version 2

Journalling Flash File System version 2 (JFFS2) is a log-structured file system optimized for flash memory devices. It has been incorporated into the Linux kernel as of release version 2.4.10, allowing for the selection of JFFS2 as the file system of choice during Linux kernel configuration. In this study, we examine the context of JFFS2 configuration during the derivation of a Linux kernel image and consider it as a hypothetical software product line (SPL) from which various JFFS2 configurations can be generated. Through the use of decision table (DT) framework, we demonstrate how to model the necessary knowledge for deriving JFFS2 configurations and how the representation of the connection between possible feature value combinations and variation activation methods can provide valuable aid for configuration derivation. Furthermore, we illustrate how this knowledge can be verified and validated through the application of prevention, detection and fixing criteria on DTs and the knowledge modeled through them. Figure 2 illustrates the feature tree of JFFS2, serving as the starting point for our running example.

3 DESIGN OF DECISION TABLES FOR SPL

In this section, we will delve into the process of designing decision tables (DTs) to support the decisions related to the derivation of final products in a software product line (SPL). As previously stated in Section 2.1, the structure of a DT is composed of four quadrants: condition subjects, conditional states, action subjects, and action values. To effectively use DTs in the context of SPLs, we must take into account certain assumptions that align DT concepts with those of SPLs. For example, we can design one DT for each software asset that can potentially be specified in multiple final products. Then, for each DT we design, we can map each element of the DT to the relevant details of the SPL. This includes:

- Mapping each *condition subject* to a *feature (variation point)*
- Mapping each *conditional state* to a *variant* of a specific feature
- Mapping each *action* to an implementation of a *variation mechanism*
- Mapping each *action value* to a *relationship* between the combinations of feature variants and the employed variation mechanisms.

A generic schema of a DT created to model the decisions about a specific asset is provided in Fig.3. In the first quadrant (top-left) there are two condition subjects, which refer to two different features of the software asset. In the second quadrant (top-right), there is information about the possible values for each specific feature. In the third quadrant (bottom-left), there is a listing of actions that represent the means to activate one or more variation mechanisms. In the fourth quadrant (bottom-right), there is a link between combinations of feature values and the actions listed in the third quadrant.

Therefore, in accordance with the principles of DTs and in line with the SPL context, the choice of a set of feature variants determines the execution of a set of variation mechanisms which can appropriately derive a final software product. In particular, let's consider these phases of a SPL:

- *domain analysis*, aimed at identifying and managing the features that the target product will have to meet;
- *variability injection*, aimed at incorporating the necessary variation mechanisms into the product design to transform it into the target product.

Each of these phases contributes to the design of specific quadrants of a DT as follows:

- domain analysis →First and second quadrants
- variability injection →Third and fourth quadrants

Each of these will be described in more detail in the following sub-sections

3.1 Domain Analysis →First and Second Quadrants

Software assets, primarily source code, form the building blocks for developers to implement specific

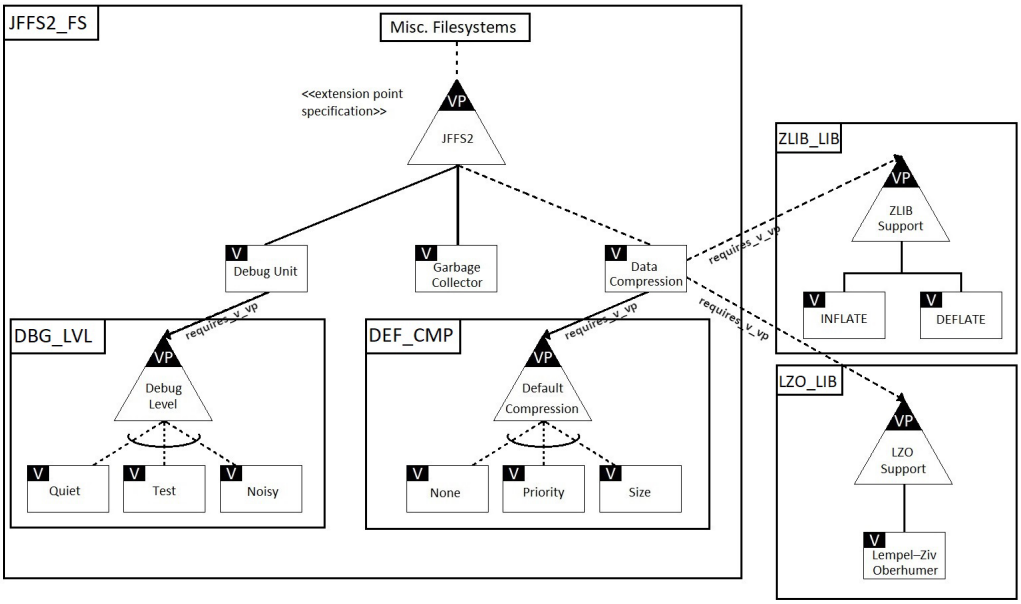

Figure 2: Feature Tree of JFFS2 SPL.

C1. Feature 1	F1 - V1			F1 - V2		
C2. Feature 2	F2 - V1	F2 - V2	F2 - V3	F2 - V1	F2 - V2	F2 - V3
A1. Variation mechanism 1	x	-	-	x	-	-
A2. Variation mechanism 2	-	x	-	-	x	-
A3. Variation mechanism 3	-	-	x	-	-	x
A4. Variation mechanism 4	-	-	-	x	-	-
A5. Variation mechanism 5	-	-	-	-	x	-
A6. Variation mechanism 6	-	-	-	-	-	x

Figure 3: A generic schema of decision table for an asset.

functionality within a software product. These assets must adhere to the domain requirements and guidelines established by the SPL architecture. Therefore, it is crucial to first derive a requirements document from the SPL domain, outlining the expected variability and commonalities in products derived through the SPL. This document serves as a blueprint for understanding the rules that each individual asset must comply with.

Once the requirements document is derived, it is necessary to clearly define the conditions that must be met by each software asset. The requirements document focuses on variability, thus, to manage variability within a single component, it is possible to represent variation points and variants through the use of a graphical notation, called the *feature model*, and to focus on the specific asset. One of the most widely used feature models is the *feature tree*, which represents variability and relations between different variants using a tree with nodes representing variants and edges representing relations. In this way, it is possible to have a simple and concise representation of conditions.

The main objective of this section is to explain how to define conditions and feature values starting from a feature tree. Specifically, we will demonstrate how to extract modeled features and insert them as conditions in the first quadrant of the DT. We will also show how to extract values for each feature and insert them in the second quadrant of the DT, according to the graphical notation used to model each specific feature value, as this notation defines the relationship between features and between a feature and its possible values.

- The *set of condition subjects* FC=$\{FC_i\}$ (i=1...n) represent the collection of features (variation points) that can be covered by the asset under consideration when it is employed in the development of a product. To gain a comprehensive understanding of feature knowledge, we can also consider the set of feature domains FD=$\{FD_i\}$ (i= 1...n) where FD_i represents the domain of FC_i, or the set of all possible values for feature FC_i.
- The *set of conditional states* FV=$\{FV_i\}$ (i=1...n) represents the set of values considered for a specific feature, as determined by selecting the appropriate values from the feature domain. Thus, FV_i=$\{F_ik\}$ (k=1...m_i) is an ordered set of n_i feature values F_ik.

Running Example: JFFS2. To support our guide Fig. 4 shows the first and the second quadrants for the example of JFFS2, they are built from feature models in Fig. 2 and according to the pattern just presented. To further illustrate our guide, Figure 4 presents the

C1. JFSS2a - Debug Unit + Garbage Collector	Debug Unit + Garbage Collector																	
C2. JFSS2b - Data Compression	Y									N								
C3. Debug Level	Quiet			Test			Noisy			Quiet			Test			Noisy		
C4. Debug Compression	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size
A1. **invoke**(JFFS2(MTD))	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A2. **invoke**(JFFS2(CRC))	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A3. **invoke**(JFFS2(Garbage Collector))	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A4. **invoke**(JFFS2(Debug Unit))	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A5. **invoke**(JFFS2(Data Compression))	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
A6. **merge**(DBG_LVL) **into** (JFFS2_FS)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A7. **VariationPoint** Debug Level **requires param**(0)	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-
A8. **invoke**(Debug Level(Test))	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-
A9. **VariationPoint** Debug Level **requires param**(1)	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-
A10. **invoke**(JFFS2(MTD))	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-
A11. **VariationPoint** Debug Level **requires param**(2)	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x
A12. **invoke**(JFFS2(MTD))	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x
A13. **merge**(DEF_CMP) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
A14. **invoke**(Default Compression(none))	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-
A15. **invoke**(Default Compression(Priority))	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-
A16. **invoke**(Default Compression(Size))	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x
A17. **deploy library** (zlib_lib) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A18. **deploy library** (lzo_lib) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Figure 4: Decision tables supporting a SPL for JFFS2.

first and second quadrants of a decision table example for JFFS2. These quadrants are constructed from the feature models in Figure 2 and in accordance with the guidelines presented in this section.

3.2 Variability Injection →Third and Fourth Quadrants

To effectively implement variability into the final product, it is crucial to incorporate it into the product design process through the use of variation mechanisms. These mechanisms are essential in determining the actions that will be performed by the components responsible for implementing the product's functionality. To achieve this, variation mechanisms must be utilized to derive variants by leveraging domain knowledge, as the actions are closely tied to product requirements. In this section, we will detail how to populate the third and fourth quadrants of a decision table (DT), using the guidelines established in the previous subsection to populate the first two quadrants, based on information derived from each domain. As previously mentioned at the beginning of Section 3, the third quadrant of a DT must contain all necessary actions, which can be considered in the context of SPLs as the appropriate combinations of variation mechanisms. Specifically, this quadrant should provide a complete list of activation methods for each specific variation mechanism. The main objectives for this quadrant are to display the various methods that can be used to activate a particular variation mechanism and to indicate how a row in this part of the DT should be set. The fourth quadrant of the DT represents the outcome of connecting the interpretation of the feature model, represented as a combination of feature values, to the actions defined in the third quadrant.

- The *set of action subjects* VM=$\{VM_j\}$ (j=1...t) represents all the potential actions that can be taken to implement a specific functionality.
- The *action values* V=$\{V_j\}$ (j =1...t) are used to connect each feature set to the corresponding action subjects. Each value, V_j=$\{$true (x), false (-), null (.)$\}$ represents the set of all possible values of action subject VM_i (with the default value set to null).

Once the DT is created, a function can be defined that maps every combination of features to a unique variation mechanism configuration using the Cartesian product of the conditional states $\{FV_i\}$ and the Cartesian product of the action values $\{V_j\}$. This function ensures that the completeness criterion of conditions and the exclusivity criterion of actions are met. Therefore, each action entry in the DT corresponds to a decision rule. Formally, the function is defined as follows:

DT: FC_1 x FC_2 x ... x FC_n →V_1 x V_2 x ... x V_t

Running Example: JFFS2. Fig.4 provides the third and the fourth quadrants of the JFFS2 example.

4 ANOMALIES HANDLING IN DECISION TABLES FOR SPL

According to classification in (Preece and Shinghal, 1994) we reformulate it from the perspective of SPLs. Fig.5 shows the description of each specific anomaly grouped into categories Redundancy, Inconsistency

and Deficiency. Each description of an anomaly has been adapted to the SPLs context, by considering a rule as the connection between the combination of feature values and the methods used to activate one or more variation mechanisms.

	CLASSIFICATION	
	Anomaly	**Description**
Redundancy	Redundant rule	A rule is the combination of two other rules.
	Subsumed rule	Two different rules contain actions that activate the same variation mechanism, but with one of them contains additional feature values.
	Duplicated rule	Two different rules involve in the same manner features and the activation of variation mechanisms.
	Unfirable variation mechanism	A rule contains a combination of the feature values that is inconsistent. So, the activated variation mechanisms are unfirable.
	Unnecessary feature combination	Neverthless the value of a feature, the activated variation michanisms are the same. Therefore, such feature can be considered redundant.
Inconsistency	Ambivalent rules	Two different rules with two or more premises containing same feature values, but leading to the activation of different variation mechanims.
	Conflicting rules	Two different rules with two or more premises containing same feature values, but leading to the activation of a contradictory set of variation mechanims.
Deficiency	Unused feature values	The combination of feature values never occur as premises. This means that a number of rules may be missing.
	Unusable variation mechanism	Some variation mechanisms are never activated by actions defined in rules.

Figure 5: Definitions of the types of anomaly.

In section 2.1 we mentioned that it is crucial for DTs to be consistent, complete, and non-redundant to avoid inconsistencies or problems in the product derivation process. By utilizing the structured nature of DTs, it is possible to easily verify and validate the set of rules derived from them. If any anomalies are found, specific mechanisms can be employed to fix them. This ensures that the knowledge modeled through the DT framework is correct and efficient. The validation provided by DTs can also be applied to the context of product derivation in SPLs by considering each condition subject as a feature, each conditional state as a variant, and each action as the set of methods used to activate a variation mechanism. This allows for easy modeling of SPL context knowledge through DTs and manual or automatic validation.

4.1 Anomalies Detection in SPLs Context

Now that we have a complete knowledge about the anomalies that may occur in modeling information through DTs applied to the SPLs context, we can describe the two steps necessary to guarantee that the information could be consistent, complete and non-redundant. The first step is the detection of anomalies in the modeled decisions. As we described in the section 2.1, the detection of anomalies can be carried out automatically or manually. This activity requires as input the set of rules derived by a DT and the criteria useful to detect the anomalies. The execution of the detection process consists in checking the anomalies existing in the provided rules, even if some anomalies may be prevented when the DT is built and populated. The goals of this process are to establish the existence of anomalies and to identify the type of problems and their source, which may be the combination of feature values, the methods used to activate the variation mechanisms or the connection between feature values and the triggered variation mechanisms. The ”detection” part of Fig.6 provides for each anomaly:

- The *Automated Checker*. a description of the operational steps used to detect the existence of a specific type of anomaly.
- The *Alert*. Information about the level of criticality of each anomaly. High critical anomalies imply that DT cannot be consulted or that they may led to incorrect results; on the contrary, low critical anomalies may led to tolerable inefficiency or they may suggest potential problems. For this reason, a classification of criticality levels is here proposed by authors. In particular we can have:
  - *Error*: anomalies that may imply inefficiencies and a wrong outcome.
  - *Warning*: anomalies that may imply inefficiencies but no wrong outcome.
  - *Notification*: potential anomalies that need to be checked by the decision makers.

Running Example: JFFS2. Fig.7 highlights the anomalies detected by the DT on the JFFS2 example. In particular, the table discovers 11 anomalies: 3 unfirable variation mechanisms (deficiency anomalies), 6 conflicting rules (inconsistency anomalies) and 2 unnecessary feature combinations (redundancy anomalies). Such anomalies will be fixed using the appropriate intervention described in Fig.6.

		DETECTION		FIXING	
	Anomaly	Automated Check	Alert	Focus	Fixing Procedure
Redundancy	Redundant rule	The definition of a decision table only allows for tables where every possible case is included in one (completeness criterion) and only one (exclusivity criterion) column. The exclusivity criterion enables the prevention of duplicated and subsumed column pairs.	Warning	Rule	The rule can be removed.
	Subsumed rule				
	Duplicated rule				
	Unfirable variation mechanism	Check on a column level: searching for a forbidden/impossible combinations of feature values (conditional states). Such combinations make the referred mechanim activation unfirable.	Error	Variation Mechanism	Unmark the cell wich links the feature values to the variation mechanism activation.
	Unnecessary feature combination	A contracted decision table, which is obtained by merging neighboring column with identical activation methods, will show a "-" entry for all condition entries related to an unnecessary feature.	Warning	Feature	The irrilevant condition subject can be removed.
Inconsistency	Ambivalent rules	Check on a column level: searching for columns which have more than one "X". It means an ambivalent activation of a variation mechanim.	Notification	Variation Mechanism	If necessary, removal of the ambivalent "X".
	Conflicting rules	Check on a column level: searching for columns which have at least a pair of "X" related to a pair of mechanisms activation in conflict.	Error	Variation Mechanism	Removal of the conflicting "X".
Deficiency	Unused feature values	Check on a column level: searching for columns which have no "X" entry. Such columns highlight the missing of a rule.	Error	Feature	Mark one or more cells of the blank column (for introducing the appropriate rule).
	Unusable variation mechanism	Check on a row level: searching for rows which have no "X" entry. A blank row makes the mechanisms activation unusable.	Warning	Variation Mechanism	Mark one or more cells of the blank row (for introducing the appropriate rule).

Figure 6: Detection and fixing procedures of anomalies in decision tables.

4.2 Anomalies Fixing in SPLs Context

Once the anomalies have been detected by the checkers, it is necessary to apply the appropriate fixing intervention according to the type of anomaly. The fixing interventions may vary according to the source and the criticality of the issue. The fixing of anomalies could also be carried out manually or automatically. Even in this case, to execute this process some fixing criteria must be defined. Then, all the detected anomalies will be treated differently according to the type of the issue and what the related fixing approach provides for. The ”fixing” part of Fig.6 describes each fixing approach adapted by considering the SPLs context. For each fixing intervention, the following information is specified:

- The *elements involved in the anomaly*, such as rules, feature value combinations and methods for variation mechanisms activation.
- The *most appropriate solution* to fix the anomaly.

This could be for example removing/refactoring rules, feature model, feature combinations and mechanisms activation. The goals of the fixing activity are to provide the set of decision without any anomaly and to define the appropriate methods to avoid the recurring of the same or similar issues.

Running Example: JFFS2. Fig.8 shows the fixed DT of the JFFS2 example. Such a table has been fixed using the procedures described in Fig.6, it does not exhibit anomalies and it appears more compact (due to the redundancy fixing intervention). Furthermore, starting from this table engineers can easily extract and use a set of business rules with no redundancy, inconsistency and deficiency.

5 THREATS TO VALIDITY

The effective and efficient use of DTs as a tool for managing software SPLs can be hindered by several threats. These include:

1. *Incorrect construction of DTs from an existing SPL*: mapping the concepts of an SPL to the elements of a DT is a complex task and an inconsistent construction of the tables can compromise the effectiveness of the tool from the start.
2. *Inadequate maintenance of DTs*: the DTs reflect the SPL, they require maintenance that aligns with the evolution of the SPL. In cases of large and complex DTs, maintenance can be challenging and prone to errors.
3. *Scalability issues*: the more variation points in an SPL, the larger and more complex the DTs will be. n this case, we can divide the threats into 2 subcategories: intra-tabular threats and inter-tabular ones.

C1. JFSS2a - Debug Unit + Garbage Collector	Debug Unit + Garbage Collector																	
C2. JFSS2b - Data Compression	Y									N								
C3. Debug Level	Quiet			Test			Noisy			Quiet			Test			Noisy		
C4. Debug Compression	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size	None	Priority	Size
A1. **invoke**(JFFS2(MTD))	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A2. **invoke**(JFFS2(CRC))	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A3. **invoke**(JFFS2(Garbage Collector))	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A4. **invoke**(JFFS2(Debug Unit))	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A5. **invoke**(JFFS2(Data Compression))	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
A6. **merge**(DBG_LVL) **into** (JFFS2_FS)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A7. **VariationPoint** Debug Level **requires param**(0)	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-
A8. **invoke**(Debug Level(Test))	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-
A9. **VariationPoint** Debug Level **requires param**(1)	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-
A10. **invoke**(JFFS2(MTD))	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x	-	-	-
A11. **VariationPoint** Debug Level **requires param**(2)	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x
A12. **invoke**(JFFS2(MTD))	-	-	-	-	-	-	x	x	x	-	-	-	-	-	-	x	x	x
A13. **merge**(DEF_CMP) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-
A14. **invoke**(Default Compression(none))	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-
A15. **invoke**(Default Compression(Priority))	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-
A16. **invoke**(Default Compression(Size))	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x	-	-	x
A17. **deploy library** (zlib_lib) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
A18. **deploy library** (lzo_lib) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

DEFICENCY: unfirable variation mechanism

INCONSISTENCY: conflicting rules

REDUNDANCY: unnecessary feature combinations

Figure 7: Anomaly detection for JFFS2.

3.1. *Intra-tabular threats*: they arise when dealing with decision tables (DTs) in software product lines (SPLs) that have high variability. The more variation points in an SPL, the larger and more complex the DTs become. In such cases, tables can have tens of thousands of columns, making them impractical for decision-makers to edit, maintain, consult, validate, and correct.

3.2. *Inter-tabular threats*: they arise when designing DTs for industrial products with hundreds or thousands of assets and hundreds of features. If one DT needs to be created for each single asset, then we will end up with as many DTs as assets and as many columns as features exhibited. In such cases, there might be a lack of anomaly investigation because the potential anomalies attributable to inter-tabular relationships are not easily identified.

To address these challenges, the authors are developing a software tool that according to the procedure described in the previous sections:

- Support and guide the user towards the construction of the tables so that the correct mapping between the SPL concepts and the constituent elements of the DT is guaranteed, in this way you can avoid the pitfalls of bad translations and associations of concepts that could undermine the correctness of the target table. (Threat 1)
- Support and guide the user during the maintenance of the DTs. The tool starting from the new aspects introduced in the SPL:
  - It suggests the elements of the table to introduce (Threat 2)
  - Supports the introduction of new rules (Threat 2)
  - Recalculate and redraw the DT with the new row and column configuration (Threat 2)
- Support the automatic (or guided) removal of anomalies so that the user does not have to deal with the direct modification of rows and columns which in some complex cases can even be several thousand. (Threat 2 and Threat 3.1 and Threat 3.2)
- Suggest to the user the best permutation of rows and columns for the DTs in order to obtain the most compact version of the DT possible. (Threat 3.1)
- Through the execution of the ”compact” functionality, consistent with the decision space described by the table, it merges the ”conditional states” and the related rules in such a way as to produce the DT configuration as compact as possible. (Threat 3.1)
- Support in a totally automatic way the verification and validation operations in order to search for all the anomalies present even in cases of very large DTs. (Threat 3.1)
- Automatically search for any presences of low cohesion between groups of decisions in the target table and then automatically (or in a guided way) break down this table into a set of sub-tables loosely connected to each other. Usually, a DT

C1. JFSS2a - Debug Unit + Garbage Collector	Debug Unit + Garbage Collector											
C2. JFSS2b - Data Compression	Y									N		
C3. Debug Level	Quiet			Test			Noisy			Quiet	Test	Noisy
C4. Debug Compression	None	Priority	Size	None	Priority	Size	None	Priority	Size	-	-	-
A1. **invoke**(JFFS2(MTD))	x	x	x	x	x	x	x	x	x	x	x	x
A2. **invoke**(JFFS2(CRC))	x	x	x	x	x	x	x	x	x	x	x	x
A3. **invoke**(JFFS2(Garbage Collector))	x	x	x	x	x	x	x	x	x	x	x	x
A4. **invoke**(JFFS2(Debug Unit))	x	x	x	x	x	x	x	x	x	x	x	x
A5. **invoke**(JFFS2(Data Compression))	x	x	x	x	x	x	x	x	x	-	-	-
A6. **merge**(DBG_LVL) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	x	x	x
A7. **VariationPoint** Debug Level **requires param**(0)	x	x	x	-	-	-	-	-	-	x	-	-
A8. **invoke**(Debug Level(Test))	x	x	x	-	-	-	-	-	-	x	-	-
A9. **VariationPoint** Debug Level **requires param**(1)	-	-	-	x	x	x	-	-	-	-	x	-
A10. **invoke**(JFFS2(MTD))	-	-	-	x	x	x	-	-	-	-	x	-
A11. **VariationPoint** Debug Level **requires param**(2)	-	-	-	-	-	-	x	x	x	-	-	x
A12. **invoke**(JFFS2(MTD))	-	-	-	-	-	-	x	x	x	-	-	x
A13. **merge**(DEF_CMP) **into** (JFFS2_FS)	x	x	x	x	x	x	x	x	x	-	-	-
A14. **invoke**(Default Compression(none))	x	-	-	x	-	-	x	-	-	-	-	-
A15. **invoke**(Default Compression(Priority))	-	x	-	-	x	-	-	x	-	-	-	-
A16. **invoke**(Default Compression(Size))	-	-	x	-	-	x	-	-	x	-	-	-
A17. **deploy library** (zlib_lib) **into** (JFFS2_FS)	-	x	x	-	x	x	-	x	x	-	-	-
A18. **deploy library** (lzo_lib) **into** (JFFS2_FS)	-	x	x	-	x	x	-	x	x	-	-	-
	1	2	3	4	5	6	7	8	9	10	11	12

Figure 8: Fixed decision table for JFFS2.

of considerable size can be represented through a ”network” of DTs, each of much smaller dimensions (Threat 3.1).

- In the case of a network of DTs, the software searches for potential anomalies related to inter-tabular relationships. Currently, the authors are studying the most appropriate checkers to support this task (Threat 3.2). In this sense, the authors

6 CONCLUSION AND FUTURE WORKS

This paper has highlighted the potential of decision trees (DTs) in aiding domain analysis and product derivation for software product lines (SPLs). We have shown that by using DTs to model complex decision scenarios and appropriate assumptions, software engineers can understand how to implement variability in the final product while adhering to requirements and constraints expressed through combinations of feature values and variation mechanisms.

DTs have also been used to model all necessary information and clearly illustrate the connections between feature values and variation mechanisms activation. This approach is particularly useful for software engineers and experts who may require knowledge that is not explicitly defined. Additionally, we have demonstrated that the assumptions made for knowledge modeling through DTs can ensure correctness, completeness, consistency, and non-redundancy of all required information, even in the context of SPLs. This is due to the adaptability of anomaly prevention, detection, and fixing criteria to the context at hand, allowing software engineers to conduct verification and validation processes on the modeled knowledge using DTs.

Looking ahead, several areas could be further explored to enhance the proposed approach. One such area is the use of SAT solvers or other similar methods to improve the effectiveness and efficiency of anomaly investigation conducted by DTs, particularly with respect to inter-tabular anomalies. Integrating these approaches could help overcome scalability issues that may arise when dealing with large and complex SPLs.

Another important direction for future research is empirical validation of the proposed approach. While the results of this study are promising, further investigation is needed to assess the effectiveness and generalizability of the proposed approach in different contexts and scenarios.

Furthermore, the completion of the software tool being developed by the authors could further enhance the proposed approach by automating and streamlining some of the more trivial and repetitive tasks involved in the process. This would allow software engineers to focus on more critical analysis and design activities related to SPL features and final products.

Finally, investigating the roles and interactions of

other actors, such as project managers and stakeholders, in the context of SPLs would be valuable in further improving the proposed approach.

In conclusion, we believe that this paper can serve as a valuable reference for future research on SPLs and the use of DTs to help software engineers manage and validate the necessary knowledge for the product derivation process.

REFERENCES

Ardimento, P., Boffoli, N., Castelluccia, D., and Scalera, M. (2016). How to face anomalies in your flexible business process? the decision table rules! In Agrifoglio, R., Caporarello, L., Magni, M., and Za, S., editors, *Re-shaping Organizations through Digital and Social Innovation*, pages 193–204. LUISS University Press - Pola Srl.

Batory, D., Benavides, D., and Ruiz-Cortes, A. (2006). Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47.

Benavides, D., Felfernig, A., Galindo, J. A., and Reinfrank, F. (2013). Automated analysis in feature modelling and product configuration. In *International conference on software reuse*, pages 160–175. Springer.

Boffoli, N., Castelluccia, D., and Visaggio, G. (2013). Tabularizing the business knowledge: modeling, maintenance and validation. In *Organizational Change and Information Systems*, pages 471–479. Springer.

Boffoli, N., Castelluccia, D., and Visaggio, G. (2014). Tabularizing the business knowledge: automated detection and fixing of anomalies. In *Information Systems, Management, Organization and Control*, pages 243–251. Springer.

Clements, P. and Northrop, L. (2001). *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional.

Czarnecki, K. and Kim, C. H. P. (2005). Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20. ACM San Diego, California, USA.

Drave, I., Kautz, O., Michael, J., and Rumpe, B. (2019). Semantic evolution analysis of feature models. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, pages 245–255.

Galindo, J. A. and Benavides, D. (2020). A python framework for the automated analysis of feature models: A first step to integrate community efforts. In *Proceedings of the 24th ACM International Systems and Software Product Line Conference-Volume B*, pages 52–55.

Horcas, J.-M., Galindo, J. A., Heradio, R., Fernandez-Amoros, D., and Benavides, D. (2021). Monte carlo tree search for feature model analyses: a general framework for decision-making. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A*, pages 190–201.

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

Krueger, C. W. (2006). New methods in software product line development. In *SPLC*, volume 6, pages 95–102.

Le, V.-M., Felfernig, A., Uta, M., Benavides, D., Galindo, J., and Tran, T. N. T. (2021). Directdebug: Automated testing and debugging of feature models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 81–85.

Maes, R. and Van Dijk, J. (1988). On the role of ambiguity and incompleteness in the design of decision tables and rule-based systems. *The Computer Journal*, 31(6):481–489.

Mannion, M. (2002). Using first-order logic for product line model validation. In *International Conference on Software Product Lines*, pages 176–187. Springer.

Maßen, T. v. d. and Lichter, H. (2003). Requiline: A requirements engineering tool for software product lines. In *International Workshop on Software Product-Family Engineering*, pages 168–180. Springer.

Pohl, K., Böckle, G., and Van Der Linden, F. (2005). *Software product line engineering: foundations, principles, and techniques*, volume 1. Springer.

Preece, A. D. and Shinghal, R. (1994). Foundation and application of knowledge base verification. *International journal of intelligent Systems*, 9(8):683–701.

Trinidad, P., Benavides, D., and Cortés, A. R. (2006). Isolated features detection in feature models. In *CAiSE Forum*, page 26.

Van Deursen, A. and Klint, P. (2002). Domain-specific language design requires feature descriptions. *Journal of computing and information technology*, 10(1):1–17.

Vanthienen, J., Mues, C., Wets, G., and Delaere, K. (1998). A tool-supported approach to inter-tabular verification. *Expert systems with applications*, 15(3-4):277–285.

von der Maßen, T. and Lichter, H. (2004). Deficiencies in feature models. In *workshop on software variability management for product derivation-towards tool support*, volume 44, page 21.

Wang, H., Li, Y. F., Sun, J., Zhang, H., and Pan, J. (2005). A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, page 46.

Zhang, W., Zhao, H., and Mei, H. (2004). A propositional logic-based method for verification of feature models. In *International Conference on Formal Engineering Methods*, pages 115–130. Springer.