Inspección de Modelos de Características: Aprovechando Patrones de Defectos y Asistencia de IA

Juan Eduardo Durán[0000-0003-1900-096X]

Universidad Nacional de Córdoba, Facultad de Matemática, Astronomía, Física y Computación

juan.duran@unc.edu.ar

Resumen. Los modelos de características (MC) son esenciales para la variabilidad en líneas de productos de software (LPS), haciendo crucial el análisis de defectos. Las inspecciones manuales son clave para problemas de dominio, requisitos y defectos no sintácticos. Las listas de verificación ofrecen inspecciones sistemáticas, reproducibles, y menos dependientes de la experiencia del inspector. Los métodos existentes para MC se basan en clasificaciones de defectos de requisitos o tipos de defectos fáciles de recordar, referenciando modelos o documentos externos. Ellos omiten patrones de defectos no obvios o fáciles de olvidar y son inaccesibles en entornos con recursos limitados. Este estudio propone una inspección de MC independiente de modelos externos, integrando clasificaciones de la literatura con patrones de defectos, centrándose en características de comportamiento. Empleamos modelos de lenguaje grandes (MLG) para mejorar la eficiencia y precisión de la inspección. Presentamos un árbol de clasificación de defectos derivado empíricamente. Los resultados empíricos de nuestro enfoque indican para los patrones de defectos que el 23% fueron no obvios y el 23% fáciles de olvidar. Los patrones no-obvios ayudaron a detectar el 60% de los defectos de ambigüedad, el 14.5% de defectos relacionados con reglas de negocio y el 11.7% de defectos de incompletitud. Los MLG contribuyeron a identificar entre el 15.5% y el 59.4% del total de los defectos. Nuestro enfoque es accesible para equipos con recursos limitados y adecuado para entornos ágiles, diseño temprano, educación e industrias complejas, uniendo teoría y práctica y estableciendo la inspección asistida por IA como una herramienta valiosa.

Palabras clave: Inspección basada en listas de verificación, Modelos de características, Modelos de lenguaje grandes, Líneas de productos de software.

Feature Model Inspection: Leveraging Defect Patterns and AI Assistance

Abstract. Feature models (FM) are vital for representing variability in software product lines (PL), making defect analysis critical. Manual inspections are essential for finding issues tied to domain understanding, requirements organization, and defects beyond syntactic checks. Checklist-based inspections are systematic,

reproducible, and less reliant on inspector expertise. Existing methods for FM inspections typically rely on requirements-based defect classifications or easily recalled defect types, often referencing supplementary models or documents. These methods overlook non-obvious or easily forgotten defect patterns. Additionally, they lack accessibility in resource-constrained settings, with dependence on external artifacts further complicating inspections. This study proposes an inspection approach independent of external models, integrating defect classifications from literature with defect patterns. The method focuses on behavioral features and employs large language models (LLMs) to enhance inspection efficiency and accuracy. Central to this approach is a hierarchical defect classification tree, empirically derived. Empirical results for our approach indicate that 23% of defect patterns were non-obvious, while 23% were easy to forget. Non-obvious patterns helped detect 60% of ambiguity defects, 14.5% of business rule-related defects, and 11.7% of incompleteness defects. LLMs contributed to 15.5%–59.4% of total defects identified. Our approach is accessible to resource-limited teams. Our approach is suited for agile environments, early design stages, education, and complex industries, bridging theory and practice while establishing AI-assisted inspection as a valuable tool.

Keywords: Checklist Based Inspection, Feature Models, Large Language Models, Software Product Lines.

1 Introduction

Motivation. FMs (see (Kang et al. 1990, Czarnecki et al. 2012)) are key for representing SPL variability. Analyzing these models to identify defects is a crucial task for ensuring their accuracy and reliability.

Automated FM analysis methods (see (Benavides et al. 2010, Galindo et al. 2019, Bhushan et al. 2020)) effectively detect anomalies like dead features or invalid models. However, manual inspection is essential for uncovering subtle defects related to domain understanding and accurate requirements representation that automated tools often miss. Automated tools excel at formal inconsistency detection, while manual inspections are better for traceability, ambiguity, and completeness defects beyond syntax.

Software artifact inspections can be ad hoc or more elaborated techniques such as checklists – see (de Mello et al. 2010). Checklists offer a systematic approach reducing oversights, reproducible results for professional/academic evaluation, inspector independence (less reliance on subjective experience), and simplicity/clarity (i.e. checklists are easy to follow). We focus on guided FM inspections using structured checks, an area less explored for FMs' challenges. These challenges include ensuring semantic clarity and consistency, precisely capturing domain semantics, achieving complete domain coverage, appropriate inspecting the FM structure, identifying and comprehensively capturing all inspection goals, accommodating inspectors that are not domain experts, addressing the necessity of reasoning during inspection process.

State of the Art. Two FM inspection approaches were identified (see (Souza et al. 2013, de Mello et al. 2014)). These approaches address FMs with features of any type.

However, specializing in behavioral features, widely used and relevant, would allow more focused examination and optimized checklists.

Souza et al. (2013) use broad non-conformity categories. de Mello et al. (2014) address easily recalled specific defects and some general ones. An alternative to these approaches is to concentrate on non-obvious or easily overlooked defect patterns because they are important to improve defect identification accuracy.

Souza et al. (2013) present defect categories without specific patterns, while de Mello et al. (2014) discuss undecomposed categories or detailed patterns. There is a need to bridge this gap between theory (categories) a practice (patterns or very specific defects).

Furthermore, existing approaches (Souza et al. 2013, de Mello et al. 2014) analyze defects involving external models (e.g. domain models, external documents). Creating a domain model is complex and error-prone, necessitating its own inspection if used for FM inspection. An unreliable domain model can cause false positives in defect identification or missed critical issues during inspection. Inspecting FMs without requiring specific external models enhances agility but poses challenges due to potentially insufficient inspector domain knowledge (DK). While FM offers some domain insights, defects like incompleteness may be missed.

Several studies demonstrate the efficacy of LLMs in detecting defects in natural language requirements – NLR - (e.g. see (Fantechi et al. 2023, Mahbub et al. 2024, Luitel et al. 2024)). However, no studies have been found that apply LLMs to the detection of semantic defects in FM. Furthermore, all reviewed approaches operate autonomously, without integration as assistants in manual inspection processes. We propose that the use of LLMs as support in formal inspections could mitigate DK limitations of reviewers, guide the inspection towards critical areas of the model, and increase the efficiency and coverage of detected defects. To our knowledge this hybrid approach, while promising, has not been evaluated in literature.

Defect detection in requirements models (including FM) differs substantially from that in NLR. In NLR, natural language processing and LLM-based techniques focus on lexical or semantic ambiguity, vague expressions, narrative inconsistencies, and contextual omissions. In contrast, graphical models exhibit structural and logical defects—such as inconsistencies across model components, missing or incorrect elements, and flawed logical conditions—that require inspection efforts more focused on model topology than on textual content.

Existing approaches lack a comprehensive framework for organizing FMs across feature granularity levels (de Mello et al., 2014) or focus solely on subdomain organization (Souza et al., 2013). A more robust organizational structure, especially for behavioral features, could improve the identification of relevant and tailored defect types.

Aim and proposal. The work aims to define a FM inspection approach that: (1) is optimized for behavioral features, (2) considers an organization of different feature granularity levels, (3) addresses defect patterns ranging from those that are difficult to forget to those that are easily overlooked, (4) integrates defect pattern extraction from FM examples with requirements literature defect categories, (5) relies only on FMs, and (6) uses LLMs to enhance the inspector's DK.

Section 2 defines features at different granularity levels and the relationships between them; it also classifies behavioral features. In Section 4 to accomplish objective (4) we propose a check tree structure including defect categories, defect subcategories, and defect patterns (considering easy to forget and non-obvious ones). The check tree was developed by combining ad hoc FM defect pattern discovery with defect categories from requirements inspection literature (see Section 3). The resulting check tree can be used with FODA FMs. Our inspection process (see Section 5) stands out by not relying on models that are external to the FM itself. This process enables the utilization of LLMs to enhance the inspector's domain expertise during the inspection. The check tree and the inspection outcomes are evaluated in Section 6.

2 Basic Concepts

FODA Feature Models. A *FODA FM* (see (Kang et al. 1990)) captures features and relationships between them. A feature can be: *Mandatory* or *optional*. *Feature elements* can have the following types of relationships: *OR* (At least one of the child features must be selected for a product), *XOR* (Exactly one of the child features must be selected for a product.), *Require* (If feature A is selected in a configuration, then feature B is also selected.), *Exclude* (Features A and B cannot be in the same configuration.).

Classification of Features. *Subsystems* (Functional subsets within a larger domain, interconnected for domain objectives), *subdomains* (Divisions within a broader domain, representing specific areas of specialization.)**,** *business processes – BP-* (Behaviors involving interrelated activities or tasks for a specific objective.)**,** *activities***:** (Groupings of related tasks contributing to BP objectives. Activities can be divided into tasks.), *tasks* (Specific, discrete actions within an activity.), *functions* **(**Capabilities/responsibilities of a subsystem or entity within the domain. Functions can be associated with BP, activities, and tasks.)**,** *external system* (A system used by the domain.)

Relationships Between Features. *Decomposes into* (used to represent a feature as a grouping of other features; it applies to subsystems and subdomains.), *includes* (Inclusion of a smaller behavior within a larger behavior, where the smaller behavior helps define the larger one.), *extends* (Execution of a smaller behavior within a larger behavior, where the larger behavior can exist independently of the smaller one.), *specializes* **(**Specific behavior as a detailed version of a general behavior. It can occur between functions.)

3 Method for Building the Defect Tree

We follow the approach in (de Mello et al. 2014), which employs ad hoc inspections to identify and categorize defect situations; in our case these defect situations are considered defect patterns. This methodology ensures that the defect tree is rooted in practical realities and concrete observations. These defect patterns are supplemented by defect classifications drawn from requirements literature. This integration strengthens the robustness of the check tree by merging practical insights with theoretical frameworks.

To bridge theory and practice, we structured the defect tree into three levels:

1. *Defect Categories:* General groups of problems identified in requirements literature. These categories encapsulate concepts, such as ambiguity, incompleteness, and incorrectness. They describe defects at a high level of abstraction, and their names do not reference specific details of FM.

2. *Defect Subcategories***:** Intermediary classifications linking defect categories to specific FM elements or parts. Subcategories emerge through ad hoc inspections of FM examples.
3. *Defect Patterns***:** Structures encapsulating recurring FM defects specializing defect subcategories. They are generalizations of ad hoc FM inspection defects.

Phase 1: Method to Identify and Classify Defect Patterns.

1. Brainstorm potential defect types based on experience, not literature.
2. Analyze the FM's approach and context to grasp less obvious meanings. Annotate aspects clearly understood and ambiguities that may reveal potential defects.
3. Identify and document defect suggestions with rationale for specific examples. This step relies on discoveries made during the process and guided by prior knowledge.
4. Group the suggested defects into broader categories.
5. Generalize suggested defects into subcategories within defect categories.
6. Synthesize defect patterns by analyzing and generalizing clusters of suggested defects that exhibit common characteristics, recurring structures, or shared underlying causes within the same defect subcategory.

Phase 2: Method for Integrating Defect Patterns with Requirements Literature Defect Categories.

1. Identify relevant literature with defect lists. Include defect category lists utilized in prior requirements-level inspection studies, even if they pertain to models types other than feature models.
2. Exclude defects inapplicable to behavioral FMs, defects unrelated to the requirements, or defects concerning optionality, variability, or dependency constraints.
3. Consolidate selected defect categories into a unified list. Broader categories should subsume more specific ones; retain only one instance of duplicated categories; resolve overlaps between categories to ensure consistency.
4. Map literature defect categories to ad hoc inspection-identified defect subcategories (where applicable). Defect patterns were derived from these subcategories.
5. Refinement of Non-Ad Hoc Categories: A. Conduct supplementary inspections to identify new subcategories and patterns for defect categories not addressed in the initial ad hoc inspections. B. Finalize the classification by incorporating insights gained through these additional inspections.

4 Applying the Method for Defect Check Tree Construction

Following Phase 1: Finding Suggested Errors and Defect Patterns. The author of the paper inspected 6 FM examples **(**e-shop, webmail, ERP, banking system, mobile media, and Go Phone**)** to find suggested errors and defect patterns.

Initiating Phase 2: Defect Category Extraction from Inspection Literature. Smarty Check inspection approach identifies 13 defect types (Geraldi et al. 2015) from a mapping study (Oliveira et al. 2017). Lamsweerde (van Lamsweerde et al. 2009) proposed a non-conformity classification for requirement specifications that was adapted for feature specifications in (Souza et al. 2013). FM Check approach (de Mello et al. 2014) uses Travassos' (Travassos et al. 2001) defect type classification adapted for FMs.

Integrating classifications from Souza et al. (2013), de Mello et al. (2014), and Geraldi et al. (2015) through filtering and consolidation, we identified eight defect categories: *Incompleteness* (Missing domain-specific information in the FM.), *Ambiguity* (FM components allowing multiple interpretations, compromising clarity*.), Incorrectness or Inadequacy* (Incorrectly or inappropriately defined FM portions, unjustified components, or those outside the intended domain.), *Incomprehensibility or Unintelligibility* (FM portions articulated incomprehensibly to stakeholders.) *Disorganization or Poor Structuring* (Feature arrangement hindering readability. Also includes cases where feature placement leads to loss of information.), Unnecessary Information or Over-Specification (FM provides excessive detail beyond what is necessary, such as information intended for later phases of development, premature decision-making, or the inclusion of duplicate elements.) *Business Rule Defects* (Omitted, inadequate, or incorrectly modeled business rules as domain functionalities.) *Extraneous Information* (Refers to redundant elements, which may include duplicated, irrelevant, or non-domain-specific components beyond the intended scope.)

Completing Phase 2: Check Tree Development by Integrating Defect Patterns and Defect Categories. Based on the categories mentioned and the defect patterns identified in our study, a defect tree was developed and is presented in Table 1.

Table 1: The Check Tree

Category	SubCategory	Patterns
Incompleteness	Missing features	• Missing feature due to consistency problem (NO). • Missing features caused by a systematic error (EF). • Missing feature due to omission (DF). • *Intentional* deviation: dependency on/exclusion with a feature that is not present (NO).
	Missing arcs between existing features	• A feature is a part of a feature group but also needs to be represented as a sub feature in another location of the tree (NO). • A sub feature must also exist as a sub feature in another location in the tree to enhance usability (NO). • A sub feature must also exist as a sub feature in another location in the tree out of necessity (DF).

	A feature with some missing children	• Missing features needed to manage a collection (EF).
		• Missing features needed to manage an object (EF).
		• Missing features required for a task (DF).
		• Missing features required for a subsystem (DF).
		• Missing sub features for an external system (DF):
		  o Only the feature name exists for external system.
		  o Insufficient children for the external system.

	Missing subtree in the FM	• Missing parent feature and its associated children (DF). • Missing subsystem (DF). • Missing external system (DF).
Ambiguity	Ambiguous feature	• Different names for the same feature (NO). • Ambiguous feature name (DF)
	Different features with the same name.	• A feature name appears in different branches of the tree but represents distinct meanings (EF). • A feature name is a descendant of another feature with the same name (DF).
Incorrectness or inadequacy	Incorrect or inadequate feature	• Incorrect parent features (DF). • Obsolete feature (NO). • Incomplete feature name (EF).

Disorganization or poor structuring	Incorrect feature placement	• Improper vertical placement (DF).
		• Improper horizontal placement:
		  o For features using the name of the same object in the predicate (EF).
		  o For features referring to the same kind of operations (EF).
		  o For features referring to the same process (DF).
		  o For features referring to the same subsystem (DF).

	Missing parent features	• Missing parent feature where child features are present:
		  o Missing feature for process or task (DF).
		  o Missing features for subsystems (DF).
		• Missing parent feature and some missing child features (DF).

Unnecessary information or over specification	Feature with unnecessary information or over specification	• Features for later phases of the development (DF). • Feature is an over specification - i.e. unnecessary or excessive details (DF).
Business rule defects	Missing features for business rule	• Missing validation for rule compliance (EF). • Lack of automation in rule application (EF). • Absence of alerts indicating rule non-compliance (NO). • Missing application of optimization (NO).
Extraneous information	Redundant features	• Non valid feature type (DF) • Non domain specific feature (DF)

Incomprehensibility or unintelligibility	Incomprehensibility in feature name	• Deduction necessary to understand a feature name (NO).

Elements of Defect Patterns. Analyzing patterns, we found that they may include the following elements:

- *Required situation*: defects in specific model condition or structure.
- *Causes:* Actions or decisions made by model authors that result in defects.
- *Unfulfilled purposes or benefits:* Failures to achieve expected objectives or intended benefits, expanding the concept of defects beyond merely performing something incorrectly.
- *Classification-Aligned Element Impact:* Impacts on FM elements whose category is consistent with external (to the FM) refined classifications.
- *Non-Standard Element Impact:* Impacts on FM elements whose category deviates from predefined external classifications.

Pattern Classification. To evaluate the utility of a defect pattern, we consider the likelihood of it being overlooked during inspection. By analyzing the identified defect patterns, we discovered the following pattern classes:

- *Difficult-to-Forget Patterns (DF)*: Easily identifiable, directly from categories, subcategories, FM element type classifications, or explicit references to specific parts of the FM.
- *Non-Obvious (NO) pattern (also Very-Easy-to-Forget):* Hard to anticipate/detect; requires expert knowledge (training), ingenuity, or deductive/critical thinking.
- *Easy to Forget Pattern (EF)*: If not explicitly addressed, these patterns are likely to be overlooked during a less structured inspection. They involve defects related to overlooked benefits, purpose or causes. Additionally, they may reference elements not aligned with predefined classifications.

These labels (DF, EF, NO) are used in the defect tree (Table 1).

For pattern classification, we first identified difficult-to-forget patterns because they are the easiest to classify. Next, we identified non-obvious patterns as they are the hardest to anticipate. The remaining patterns are the easy-to-forget ones.

5 Proposed Method to Conduct Inspections

Roles Involved. This section details the inspector's role: using a check tree and LLM to identify FM defects and bridge knowledge gaps.

Crucial additional roles include *collector* (Consolidates identified defects.), *author* (Develops/explains the FM and corrects defects.), *organizer* (Selects inspectors/collector, distributes materials, and ensures defect correction.)

Tasks Performed by the Inspector.

1. Taking notes interpretating FM elements/sections.

2. Using the LLM clarifies FM specifics. Examples: How is the subsystem or subdomain Z of system X defined? What does feature Z of system X refer to?
3. *Check Tree Defect Detection***:** The approach involves:
   - Examine subcategories within defect categories for potential issues.
   - For each subcategory, the inspector first searches for defects independently and then uses subcategory patterns to find more. The reasons for doing this are: A. Patterns may not comprehensively cover all aspects of a subcategory. B. The inspector should develop familiarity with a subcategory before relying on its patterns.
4. *LLM defect detection* serves to identify certain types of defects, such as incompleteness, issues related to external systems, and missing business rules. The LLM can assist with specific inquiries:
   - *For detecting incompleteness:* What are the subsystems/subdomains of system X? Into which tasks/functions can subsystem/process X be decomposed? In system X, we have S. What other functions/tasks/processes might be missing in S?
   - *For detecting missing business rules:* What types of business rules are typically associated with a system/subsystem/subdomain X?
   - *For detecting issues with external systems:* Which external systems are commonly used by system X? Which external systems are typically used by subsystem Y of system X? What functions do external system Y usually offer?
   - *For identifying inaccuracies or inadequacies:* Is feature X obsolete or outdated?

For the tasks involving an LLM we utilize what's known as an *informational prompt* or *query prompt*. With this type of prompt, the user poses a clear and direct question to obtain data, explanations, or context on a specific topic. To construct a prompt, we replace the variables within a given question with the appropriate names.

6 Evaluation of Check Tree and Inspection Outcomes

Conducting Inspections. For each PL example within the dataset we considered, the author of this paper applied the inspection method (see Section 5) utilizing a Copilot LLM to augment the existing set of suggested defects used to construct the check tree.

Dataset considered. We started with domain expert-crafted FMs from the SPLOT dataset, a public repository. We selected six sufficiently large FMs predominantly containing behavioral features: e-shop, webmail, PL, ERP PL, banking system PL, mobile media PL, and Go Phone PL.

Deriving Key Values for Research Question Exploration. For each example, we counted: suggested defects (total), defects per category, defects per subcategory, and defects per pattern. Only identified cases were counted; others were zero.

We considered the following research questions to evaluate the check tree:

RQ1: Evaluating the utility of inspections for identifying defect patterns of different kinds: What percentage of defect patterns are non-obvious (very easy to forget)? What percentage of defect patterns are easy to forget?

RQ2: Assessing the impact of non-obvious and easy to forget patterns on identifying defects within their corresponding categories: What is the percentage of defects per non-obvious pattern within its defect category? What is the percentage of defects per easy to forget pattern within its defect category?

For inspection outcomes evaluation we considered the following research questions:
RQ3: Evaluating the utility of using LLMs for detecting additional defects: What is the percentage of defects discovered using an LLM?
RQ4: Measuring the impact or significance of each defect type: What is the distribution of defect types overall? *Metric*: Number of defects per defect type / total number of defects. What is the distribution of defect types by model? *Metric*: Number of defects per defect type in model M / total number of defects in model M.
RQ5: Assessing the impact of FM size on the number of defects identified: What is the defect-to-size ratio across models?

6.1 Results

RQ1: Proportion of Non-Obvious Defect Patterns: 23%.
Proportion of our non-obvious defect patterns = 17.9%. A significant portion.
Proportion of Easy to forget Defect patterns: 23 %. This is a significant proportion. The proportion of difficulty to forget defect patterns is 54%.

RQ2: Proportion of Defects Attributed to Non-Obvious Patterns Within Their Defect Categories

Table 2: Defect Attribution to Non-Obvious Patterns

Category of defects	Non-Obvious Pattern	Proportion
Incompleteness	Missing feature due to consistency problem	0.92 %
	Intentional deviation	4.6 %
	Missing arc where child in feature group	3.7%
	Missing arc for usability	1.85%
Incorrectness or inadequacy	Obsolete feature	33.3%
Business rule	Alert	9,5%
	Optimization	4,75%
Ambiguity	One feature is a descendant of the other, and both have the same name	40%
Incomprehensibility or unintelligibility	Deduction necessary to understand a feature name	100%

Table 2 results show: for ambiguity defects 40% are non-obvious patterns. For business rule defects 14.5% are non-obvious patterns. For incompleteness and omission: 11.7% of suggested defects are non-obvious patterns.

RQ2: Proportion of Defects Linked to Easy to Forget Patterns Within Their Defect Categories. Table 3 shows that 85.6% of business rule defects fall into this group, followed by 33.4 % for incorrectness or inadequacy. Additionally,16% are associated with disorganization or poor structuring, 20% with ambiguity, and 14.75% with incompleteness or omission.

Table 3: Defect Attribution to Prone to Oversight Patterns

Category of defects	Easy to forget pattern	Proportion
Incompleteness	Missing features due to systematic error	9.2%
	Missing children to manage collection	3.7%
	Missing children to manage object	1.85%
Incorrectness or Inadequacy	Incomplete feature name	33.4%
Ambiguity	Features in different branches with the same name	20%
Disorganization or poor structuring	Improper horizontal placement for features referring to the same object.	12%
	Improper horizontal placement for features referring to the same kind of operations.	4%
Business rule defects	Missing validation for rule compliance.	47.6%
	Lack of automation in rule application.	38%

RQ3: In our study we employed Microsoft Copilot for identifying additional defects. The proportion of defects identified by Copilot was as follows: 42.8% for Go Phone, 31.5% for Banking Software, 15.5% for e-shop, and 59.4% for ERP. Copilot significantly aided defect identification across all four analyzed examples. Furthermore, PLconceptual complexity correlated positively with higher defect detection rates (e.g., ERP and banking systems). Notably, many of the Go Phone defects identified by Copilot arose during efforts to modernize this PL.

RQ4: The distribution of defect types observed was as follows: incompleteness accounted for 57.1 %; non-organization or poor structuring, 13.2%; business rules, 11.1 % (as FMs rarely incorporated business rules); extraneous information, 8.4%; unnecessary information or over specification 4.7 %; ambiguity 2.6%, incorrectness or inadequacy 1.6%; and incomprehensibility or unintelligibility, 1%.

RQ4: Distribution of Defect Types by Model

Table 4: Defect Distribution by Category Across Feature Models

Category of defect	Mobile media	Web mail	ERP	e-shop	Banking software	Go-Phone
Incompleteness	75%	50%	78,4%	75%	26.3%	76%
Ambiguity	25%	8,3%		3.4%		4.7%
Incorrectness or inadequacy		8,3%				9.5%
Disorganization or poor structuring		16,6%		3.4%	33,3 %	9.5 %
Unnecessary info or overspecification				3.4%	12,2 %	

Business rule			21,6%	8,6 %	14%	
Extraneous information				13.7%	14%	
Incomprehensibility or unintelligibility		16,6%				

Table 4 indicates that incompleteness/omission defects were prevalent across all examples. Poor structuring and ambiguity defects were identified in two-thirds of the examples each. Business rule defects were significant in half the examples.

RQ5: The observed defect-to-size ratios across models were: 0.65 for ERP, 0.46 for banking software, 0.27 for e-shop, 0.27 for Go Phone, 0.15 for web mail, and 0.15 for mobile media. The defect-to-size ratio is notably higher in examples that are more complex and/or incorporate a greater number of technical concepts, such as ERP, banking software, and e-shop. However, the variation in the proportion of defects does not appear to correlate with the size of the FM.

6.2 Threats to Validity

We considered only six examples due to time constraints and data availability, potentially biasing defect patterns and missing others. Larger datasets would improve validity and pattern discovery.

Reliance on a single inspector risks experimenter bias, even with LLM assistance. Multiple inspectors with diverse expertise could improve reliability and uncover more subcategories or defect patterns.

FM author validation of suggested defects was absent. While many defect types are likely true positives, business rule/external system defects may be false positives due to context. FM author validation would improve ecological validity.

SPLOT examples' real-world representativeness and applicability are uncertain, limiting ecological validity of the findings. Future research should evaluate defect patterns in industrial settings to improve applicability.

The definition of patterns is based on our analysis of inspected FMs. More inspections of more examples could refine this definition and enhance their benefits.

7. Discussion

Addressing Documentation and Data Challenges. Our method enables FM inspection without external assumptions in scenarios with unavailable, sparse, unclear, or unreliable domain documentation—common in industrial projects. Our additional model-independence approach is effective in projects with limited or inconsistent initial data, making it valuable across development stages.

Application Scenarios. In agile environments with limited time and often incomplete domain documentation, our approach enables rapid inspections. It's applicable early in FM design, even before concrete products, and for reviewing FMs in evolving systems by focusing on new features. This method is useful in industries with complex SPLs requiring continuous validation and in educational settings for teaching effective FM inspection.

Customizable Depth and Enhanced Efficiency. The approach's graduality and customizable depth allow tailoring inspection detail, from defect categories to defect patterns. We also use LLM-suggested classifications for completeness issues to reduce manual effort, enhance efficiency and accessibility for inspectors with less DK by amplifying their analytical capacity.

Mitigating DK Gaps. With partial or general domain knowledge, the FM itself provides context for defect detection. Furthermore, an LLM can significantly mitigate any DK gaps by clarifying concepts, and feature decomposition making the inspection process more robust and accessible.

Addressing Inspector Experience Levels. Difficult to forget defect patterns aid novices unfamiliar with model element classification. Easy to forget defect patterns assist novices and intermediates by highlighting easily missed issues. Non-obvious patterns benefit all experience levels, uncovering issues often overlooked even by experts without a structured framework.

Structured vs. Ad Hoc Inspections. Although ad hoc inspections might catch some defects missed by checklists, checklists ensure consistent review of key elements, reducing critical oversights. Ad hoc inspections often lack the structure for thoroughness.

8. Conclusion

Some Limitations. The approach focuses solely on behavioral features and their organization, omitting other potentially valuable features. The manual application of the robust defect check tree may be slower and less precise than automation.

Practical Applications and Advantages. This approach is applicable at any FM development stage without extra documentation. Its ability to identify non-obvious patterns can significantly enrich future defect studies. The method facilitates early-phase inspections, providing immediate defect insights without detailed documentation.

Benefits of Achieving the Objectives. Tailoring our approach to FMs encompassing behaviors at varying granularities offers significant value to stakeholders interested in detailed operations. By not relying on separate domain or auxiliary models, our approach avoids risks associated with their quality, eliminating the need for their validation and enabling early defect detection without external model dependencies.

Future work.

- Involve multiple participants with varied expertise levels to minimize experimenter and selection biases.
- Collaborate with authors of requirements models to confirm or refine detected defects, addressing external threats such as population and ecological validity.
- Gather input from inspectors and model developers, as well as feedback from SPL experts on detected defects to assess their relevance.
- Test the method on additional feature models across different areas and domains to see if similar defects emerge.

- Evaluate the approach in industrial settings using real-world requirements models to assess practicality and value.
- Measure time and resources required for inspections and compare them to the benefits in defect reduction.

References

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S. (1990) Feature-Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, SEI 1990.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., Wąsowski, A. (2012) Cool features and tough decisions: a comparison of variability modeling approaches. In Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems, pp. 173-182.

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.

Galindo, J. A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A. M., & Ruiz-Cortés, A. (2019). Automated analysis of feature models: Quo vadis? *Computing*, *101*, 387-433.

Bhushan, M., Negi, A., Samant, P., Goel, S., Kumar, A. (2020) A classification and systematic review of product line feature model defects. *Software Quality Journal*, vol. 28, p. 1507-1550.

IEEE (2008). STD 1028-2008: IEEE Standard for Software Reviews and Audit.

de Mello, R. M., Pereira, W. M., Travassos, G. H. (2010) Activity Diagram Inspection on Requirements Specification. XXIV Simpósio Brasileiro de Engenharia de Software.

de Mello, R. M., Nogueira, E., Schots, M., Werner, C. M. L., & Travassos, G. H. (2014). Verification of Software Product Line Artefacts: A Checklist to Support Feature Model Inspections. *J. Univers. Comput. Sci.*, *20*(5), 720-745.

Souza, I. S., da Silva Gomes, G. S., Neto, P. A. D. M. S., do Carmo Machado, I., De Almeida, E. S., & de Lemos Meira, S. R. (2013). Evidence of software inspection on feature specification for software product lines. *Journal of Systems and Software*, *86*(5), 1172-1190.

Fantechi, A., Gnesi, S., & Semini, L. (2023). Rule-based NLP vs ChatGPT in ambiguity detection: a preliminary study. En A. Ferrari et al. (Eds.), *Joint Proceedings of REFSQ 2023 Workshops* (Vol. 3378). CEUR-WS.

Mahbub, T., Dghaym, D., Shankarnarayanan, A., Syed, T., Shapsough, S. Y., & Zualkernan, I. A. (2024). Can GPT-4 aid in detecting ambiguities, inconsistencies, and incompleteness in requirements analysis? A comprehensive case study. *IEEE Access, 12*, 171972-171992.

Luitel, D., Hassani, S., & Sabetzadeh, M. (2024). Improving requirements completeness: automated assistance through large language models. *Requirements Engineering, 29*(1), 73-95.

Geraldi, R. T., Oliveira Jr, E., Conte, T., & Steinmacher, I. (2015, April). Checklist-based Inspection of SMarty Variability Models. In *Proceedings of the 17th International Conference on Enterprise Information Systems-Volume 2*, pp. 268-276.

Oliveira Jr, E., Geraldi, R. T. (2017). Defect Types and Software Inspection Techniques: a Systematic Mapping Study. *J. Comput. Sci.*, *13*(10), 470-495.

Lamsweerde, A. van. (2009). *Requirements engineering: from system goals to UML models to software specifications*. John Wiley & Sons, Ltd.

Travassos, G. H. In Rocha, A. R. C., Maldonado, J. C., Weber, K. C. (2001) Qualidade de Software – Teoria e Prática. Prentice Hall.