Software Quality Journal (2024) 32:1007–1037
https://doi.org/10.1007/s11219-024-09679-z

RESEARCH

A rule-based approach for the identification of quality improvement opportunities in GRL models

Mawal A. Mohammed[1] · **Mohammad Alshayeb**[2,3] · **Jameleddine Hassine**[2,3]

Accepted: 12 May 2024 / Published online: 6 June 2024

Abstract

Goal-oriented modeling languages have been proposed to elicit, analyze, and document high-level system requirements in the early stages of the requirements engineering (RE) process. Problems during this stage may disseminate to the subsequent stages in the software development process and artifacts. Therefore, improving the quality of goal models would improve the quality of the requirements and, consequently, the quality of the developed system. This paper proposes an approach to help modelers identify quality improvement opportunities in Goal-oriented Requirements Language (GRL) goal models. To this end, a list of GRL bad smells (i.e., bad quality symptoms) is introduced and evaluated by experts. Then, an automated rule-based technique is proposed to detect the instances of these smells. The proposed approach is evaluated using a dataset gathered from academic and real-world projects. The results show that the developed technique could successfully detect the instances of the proposed bad smells in the evaluation models. We also found that the instances of the proposed bad smells were prevalent in both academic and industrial settings. The proposed bad smells and the detection technique provide a tool to locate quality improvement opportunities in GRL goal models.

Keywords Requirements engineering · Goals · Bad smells · Rule-based · OCL

✉ Mohammad Alshayeb
alshayeb@kfupm.edu.sa

Mawal A. Mohammed
maw.mohammed@psau.edu.sa

Jameleddine Hassine
jhassine@kfupm.edu.sa

[1] Department of Software Engineering, Prince Sattam Bin Abdulaziz University, Al-Kjarj, Saudi Arabia

[2] Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

[3] Interdisciplinary Research Center for Intelligent Secure Systems, 31261 Dhahran, Saudi Arabia

1 Introduction

Requirements engineering is the process of eliciting, specifying, documenting, and managing requirements of software systems (Pohl, 2010). In traditional requirements engineering practices, the focus is on modeling the functionalities of the system (DeMarco, 1979; Rumbaugh et al., 1991). A little attention is given to the justification of the specified functionalities (i.e., the rationale behind the definition of the functionalities). The concept of "goals" was introduced to link the specified system functionalities with stakeholders' business and operational objectives. The objective of these associations is to provide a tool to explore the context of the system effectively (Lapouchnian, 2005).

Goals are defined as the objectives of the stakeholders that the system is expected to achieve through the cooperation of its agents (Van Lamsweerde, 2000). Goal models are used at the early stage of the requirements engineering process to model high-level requirements. The quality of these models has a significant impact on the later stages of the software development process as problems in this stage disseminate to all subsequent stages in the software development process (Denger & Olsson, 2005; Knauss et al., 2009). Most importantly, goal models are concerned with the context of the system, and problems in specifying the context are detrimentally expensive (Tukur et al., 2021). In fact, context problems are the most expensive problems to rectify. Therefore, the quality of goal models should be at the center of the attention of the quality assurance processes and activities.

To help improve the quality of requirement artifacts, requirement engineers conduct manual and automatic reviews (Salger, 2013). Requirements formal reviews contribute effectively and positively to the success of software projects (Hofmann & Lehner, 2001). One of the objectives of the review process is to identify quality problems that need to be fixed. However, manual reviews, compared to automatic reviews, are expensive, have long feedback loops, and may be inconsistent (Femmer, 2017; Femmer et al., 2016; Katasonov & Sakkinen, 2006). To be successfully applied, manual reviews require reviewers to have a high degree of domain understanding and expertise (Salger, 2013; Zelkowitz et al., 1983). They also require the involvement of the relevant stakeholders (Salger, 2013), who have to read, understand, and review each requirement artifact. Therefore, increasing the automation of quality reviews expedites the whole review process. Besides, it increases the effectiveness and viability of manual reviews as it relieves reviewers from capturing issues that can be automatically detected, which enables them to focus on deeper issues, e.g., semantic issues, that require human intervention.

Quality standards, including ISO/IEC/IEEE—29,148–2018 (Standardization, 2018), provide an overarching view of quality. They specify characteristics of good quality, but they do not specify symptoms where these characteristics are violated. To help assure the compliance of goal-oriented requirements artifacts with the characteristics of good quality requirements, deviations from good quality characteristics should be identified. In this work, we use the concept of bad smells to articulate symptoms of bad quality. Articulation of bad smells is the first step to automate the detection of their instances. These instances represent quality improvement opportunities that can be addressed to improve the quality of the subject artifact. This approach has been used with various software artifacts, and bad smells have been articulated for various artifacts, including source code (Fowler, 2018), software models (Alkharabsheh et al., 2019; Misbhauddin & Alshayeb, 2015; Mumtaz et al., 2019), and requirements artifacts, including natural language constructs (Femmer et al., 2014), use-case models (El-Attar & Miller, 2010; Seki et al., 2019), and goal models (Asano et al., 2017; Yan, 2008).

Several goal modeling frameworks were proposed in the literature, including NFR (Chung et al., 2012; Mylopoulos et al., 1992), AGORA (Kaiya et al., 2002), KOAS (Dardenne et al., 1993; Van Lamsweerde & Letier, 2004), i* (Yu, 1997), and GRL (ITU-T, 2018). These frameworks share common concepts, including refinement and operationalization, to help express concepts that appear in the early stage of the requirements engineering process. They also offer features such as satisfaction analysis to help evaluate satisfaction levels and explore the available alternatives. In addition to requirements engineering, goal-modeling frameworks have been successfully used in many other fields, such as variability analysis, compliance policies, legal modeling, etc. (Horkoff et al., 2016). We selected GRL as our target goal modeling language as it was standardized and part of ITU-T's User Requirements Notation (URN) standard (ITU-T, 2018). The outcomes of this work tend to be more influential when GRL models are used as a core part of the requirements engineering process. Therefore, the quality of these models is the focus of this work. More particularly, the focus is on detecting structural bad quality symptoms automatically.

Given the impact of bad quality symptoms and the difficulty of manual reviews, this paper contributes to alleviating this problem as follows:

1. Introduce a catalog of 5 GRL bad smells: negligible sub-model, non-strategic sub-model, un-operationalized objective, disturbed operationalization, and unknown contribution. This catalog can be used in various ways to improve the quality of GRL models. It can be used as a basis to automate the process of the detection of bad quality symptoms. It can also be used to educate GRL modelers on such bad symptoms to avoid them.
2. Propose a rule-based detection technique to detect the instances of the proposed bad smells. The proposed approach has been evaluated using 12 case studies taken from the academic and industrial development settings.
3. Provide tool support as an extension to the jUCMNav tool (i.e., the most comprehensive GRL tool) (Amyot et al., 2011) to promote the adoption of the proposed bad smells and detection technique. The developed tool is freely available online to download and use.[1]

The rest of this paper is organized as follows: Sect. 2 presents the background of this work. The related works are presented in Sect. 3. In Sect. 4, the introduced bad smells are presented. In Sect. 5, the detection approach is presented. The evaluation of the proposed approach is presented in Sect. 7. In Sect. 8, the threats to the validity of this work are identified and discussed. Finally, in Sect. 9, conclusions and future work are presented.

2 Background

In this section, the GRL framework is introduced, and its main constructs are presented. GRL is a high-level requirements modeling and reasoning language (Chung et al., 2012; Yu & Mylopoulos, 1998). It is part of the ITU-T User Requirement Notation (URN) standard (ITU-T, 2018).

Goal models, including GRL, provide a conceptual framework that allows requirements engineers to explore the domain of the system effectively. They help express concepts that appear during high-level requirements engineering, such as why that requirement is

[1] https://github.com/MawalMohammed/GRL-bad-smells-detection-jUCMNav-extension

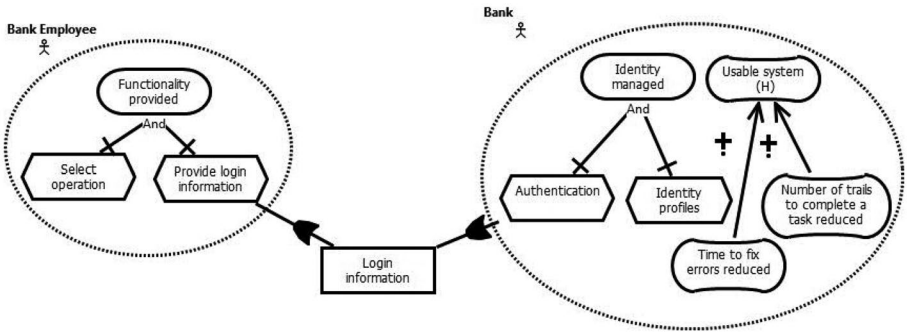

Fig. 1 GRL running example

needed and how it can be achieved (Yu, 2011). GRL provides, as well, a mechanism to reason about the structural and behavioral aspects of goal models using satisfaction analysis, which enables the evaluation of various GRL strategies with respect to the satisfaction of actors' objectives, as well as assessing the selected criteria (e.g., importance values) (Horkoff & Yu, 2013). Moreover, goal models are used to clarify requirements by elaborating on goals using refinement and decomposition (Van Lamsweerde, 2001). In addition, goal models can be used as a validation tool to validate requirements specifications by tracing back these specifications to the objectives of stakeholders (Van Lamsweerde, 2000). Despite limited adoption, GRL models are used in the industry (Yu et al., 2013). A recent survey highlighted their influence on roughly 70 patents that employ URN notation. (Amyot et al., 2022).

In what follows, we use the GRL model of Fig. 1 to introduce the different GRL constructs. For a complete description of the language and its constructs, readers are referred to the ITU-T standard (ITU-T, 2018). GRL models can be created using jUCMNav tool (Amyot et al., 2011). The experimental data, along with the examples used in this paper, are produced using this tool.

The main constructs of the GRL language (presented in Table 1) are actors, intentional elements, and links. Actors are the active entities in the domain of the system that have objectives to be satisfied. They are used to represent the relevant stakeholders in the

Table 1 Main Constructs of GRL models

Actors and elements		Links	
Actor		Dependency	
Goal		Contribution	
Softgoal		AND-Decomposition	AND
Task		XOR-Decomposition	OR
Resource		OR-Decomposition	XOR

domain or the system under development. The bank employee and the bank in Fig. 1 are examples of actors.

Goals and softgoals are used to model the objectives of the stakeholders. The difference between goals and softgoals is that goals have clear-cut criteria to measure their satisfaction while softgoals do not. "Identity managed", a functional requirement, is an example of a goal, while "Usable system", a non-functional requirements (i.e., usability), is an example of a soft goal. Tasks and resources are used to operationalize the objectives of the stakeholders. Tasks represent a course of actions to be done to satisfy (or contribute to satisfying) the respective goals/softgoals. "Provide login information" in Fig. 1 is an example of a task. Resources are elements that are needed to satisfy the respective goal/softgoal. They can be physical (e.g., hardware) or informational (e.g., data). "Login information" in Fig. 1 is an example of an informational resource.

Links are used to connect intentional elements. GRL supports three types of decomposition links: AND, OR, and XOR. For example, in Fig. 1, "Identity managed" is AND-decomposed into two tasks: "Authentication" and "Identity profiles". Moreover, elements can contribute to each other negatively or positively. To depict these relationships, contribution links are used. Contribution links are annotated with contribution types. These types can take various levels. These levels are: Make, SomePositive, Help, Unknown, Hurt, SomeNegative, and Break. For example, "Time to fix errors reduced" is "Help-contributing" to "Usable system" softgoal. Contribution links can also be annotated with quantitative values that range between -100 to 100. Actors usually depend on each other. To depict this relationship, dependency links are used. In Fig. 1, the bank actor depends on the bank employee actor in providing login information. The objectives of the actors and the mechanisms used to operationalize their objectives are represented using elements and links. In GRL, there are four types of intentional elements: goals, softgoals, tasks, and resources.

In GRL goal models, several constructs are used to represent the system's requirements at the early stage of requirements engineering. Actors are the entities that have objectives that need to be achieved by the system. Dependencies model the relationships between these actors. High-level goals and softgoals are represented within each actor, which can be further refined into lower-level goals or softgoals through decomposition or contribution links, as necessary. The lowest-level goals and softgoals are operationalized using tasks or resources to ensure their achievability. It should be noted that the development of the model is an iterative process, with modelers revisiting and refining the constructs until the model is fully developed.

3 Related work

The literature on bad smells in software development is extensive, covering various aspects of software artifacts, including code and models. While this work introduces bad smell detection in general, the focus then moves to modeling bad smells, specifically in the context of GRL models. This work builds upon and extends prior research on the identification and detection of bad smells in goal models.

3.1 Bad smell detection

Bad smells are symptoms of bad quality (Fowler, 2018). They can infect various software artifacts, including code (Fowler, 2018) and models (Mumtaz et al., 2019). For instance,

"Long parameter list" is an example of a code smell (Fowler, 2018). Having too many parameters is not an error or defect; it is a symptom of bad quality that might lead to problems later. "A state without an incoming transition" in a state diagram is an example of a model bad smell (Arendt & Taentzer, 2010). This configuration is not an error or defect. However, it is a bad quality symptom that might lead to problems later.

To detect bad smells, several techniques have been proposed, including rule-based techniques, search-based techniques, machine learning techniques, metric-based techniques, etc. (AbuHassan et al., 2021; Alkharabsheh et al., 2019; Baqais & Alshayeb, 2020; Misbhauddin & Alshayeb, 2015).

Several techniques are employed to detect bad smells in software code and design models (Misbhauddin & Alshayeb, 2015; Sharma & Spinellis, 2018). Metrics were used to detect bad smells (Bertran, 2011; Dexun et al., 2012; Nongpong, 2015) when the addressed bad smells can be quantified. To detect bad smells using metrics, a quantifiable association between the addressed bad smell and one or more properties of the subject artifact is established. These properties include size, coupling, cohesion, etc. Metrics are used in conjunction with thresholds to specify the extreme values and, consequently, bad smells.

Machine-learning techniques were also used in bad smells detection (Fontana et al., 2016; Hozano et al., 2017; Maneerat & Muenchaisri, 2011). In this approach, the problem of bad smell detection is formulated as a classification problem. The objective is to classify software artifacts into smelly or not smelly. To prepare software artifacts for classification, the relevant features (the ones that can differentiate the smelly or not smelly artifact) are extracted from these artifacts. After that, different classification algorithms (Kotsiantis et al., 2007) can be used to detect smelly artifact.

Graph algorithms were employed to detect bad smells (Fontana et al., 2017; Nguyen et al., 2009; Peiris & Hill, 2016; Tekin & Buzluca, 2014) as well. The structure of the graph lends itself to represent various software artifacts such as code, class diagrams, etc. The basic structure of a graph consists of vertices and edges (West, 1996). To represent a software artifact as a graph, elements of the artifact are represented as vertices, and the relationship between its elements is represented as edges. Bad smells are then represented as graph substructures or properties. To locate instances of bad smells, graph algorithms, such as graph mining algorithms (Tekin & Buzluca, 2014), are applied.

Researchers also used metaheuristic search algorithms to detect bad smells (Boussaa et al., 2013; Kessentini et al., 2014; Ouni et al., 2017) in which the subject software artifact is formalized and transformed into a search space. Then, bad smells are represented as generic solutions to be searched for in the prepared search space (i.e., the subject software artifact). After that, one or more search algorithms (Lones, 2011) are used to search for specific instances of the generic solutions (i.e., bad smells). These search algorithms start from random solutions and utilize several mechanisms and heuristics in seeking optimal solutions (i.e., concrete instances of bad smells).

Rules were also employed to detect bad smells (Czibula et al., 2015; Kessentini et al., 2011; Lee & Geem, 2005), where an association between bad smells and the subject software artifact is established. This association can be direct and descriptive, or it can be indirect through proxies such as metrics. Direct rules are used to describe concrete configurations that represent bad smells. To identify instances of bad smells, the developed rules are applied to the subject software artifact.

3.2 Goal modeling bad smells

In goal models, only a few studies addressed bad smells. Asano et al. (Asano et al., 2017) investigated the quality of goal refinements in AGORA goal models (Kaiya et al., 2002). As a result, a group of four bad smells was introduced: (1) low-semantic relation: this smell is concerned with the relevancy of elements to their parent. To detect its instances, the use case-frames technique (i.e., natural language processing technique) is proposed. The concept behind this technique is to calculate the similarity between the parent and its children based on a similarity dictionary, (2) Too many siblings: this bad smell is concerned with the relevancy of elements to their parent as well. To detect its instances, the number of children metric is used with an upper threshold, (3) Too few siblings: this bad smell is concerned with the completeness of the refining a parent into children. To detect its instances, the number of children metric is used with a lower threshold, and (4) Coarse-grained leaf: this bad smell is concerned with the quality of elaboration of refinement. To detect its instances, the depth of refinement metric is used with a lower threshold.

For GRL, several guidelines for developing goal models using GRL were integrated into jUCMNav tool (Amyot et al., 2011; Yan, 2008). These guidelines are divided into several categories. Many of these categories are irrelevant to our work, such as the one that is concerned with legal compliance analysis (Ghanavati et al., 2014). Only three categories are found relevant. The first category is concerned with the completeness and consistency of GRL models. This category presents several rules that check the completeness and consistency of GRL models in various aspects. For example, one of these guidelines says that each actor should include, at least, one element whose importance value is not zero. The second category is concerned with profiling i* language using GRL (Amyot et al., 2009). Given that i* is a goal modeling language, its profile provides relevant guidelines. For example, i* profile emphasizes complete dependency relations by having depedums. While the GRL standard (ITU-T, 2018) offers various options for dependency relations, the i* profile specifically limits dependency relations to cases where the presence of a dependum is required. An extended version of i* guidelines with their violations can be treated as bad smells which can be found in iStarGuide.[2] The third category focuses on empty constructs, such as an actor with no enclosed elements. While it is not considered as an error or a defect, empty constructs are generally seen as undesirable.

Mohammed et al. (2022a, b) formalized the circular dependency bad smell and developed an approach based on the search-based simulated annealing (SA) algorithm to detect its instances in GRL models. In a more recent work, Mohammed et al. (2022a, b) introduced four structural GRL bad smells: (1) Overly Ambitious Actor bad smell: represents an actor with too many objectives (exceeding a pre-determined threshold), (2) Overly Operationalized Actor bad smell: represents an actor with too many tasks and resources compared to its low-level goals and softgoals (exceeding a preset ratio), (3) Deep Hierarchical Refinement bad smell: represents an actor with a very deep refinement hierarchy (exceeding a preset maximum chain size), and (4) Highly Coupled Element bad smell: represents an element having a large number of links (incoming or outgoing links) that exceeds a predetermined threshold. The detection of the instances of these four bad smells is based on a set of metric-based rules and was implemented in an Eclipse plugin tool, called GSDetector (GRL Smells Detector).

[2] http://istarwiki.org/tiki-index.php?page=iStarGuide

Building upon previous work that identified four bad smells in GRL, our paper contributes five additional bad smells, which we believe significantly enriches the body of knowledge in this area for several reasons: (1) By expanding the list of bad smells, practitioners now have access to a more comprehensive set of guidelines, enabling them to further improve the overall quality of their GRL models. (2) The newly proposed bad smells address specific challenges not covered by the initial four bad smells, shedding light on previously unexplored aspects of GRL and providing valuable insights to the research community. (3) Our work opens up opportunities for future research to investigate the relationships between various bad smells, their underlying causes, and their potential impact on GRL model quality. This can lead to the development of more effective tools and techniques for managing GRL bad smells, ultimately benefiting the wider GRL community.

In this paper, we start by expanding the list of GRL bad smells with 5 new bad smells. Then, we develop a technique to detect the instances of the proposed bad smells. To improve the adoption and usability of the proposed bad smells and detection technique, we extend the jUCMNav tool to accommodate the detection of the instances of the proposed bad smells.

4 The proposed GRL bad smells

The following subsection introduces the proposed bad smells. These smells are cataloged using the following items:

- A brief description of the proposed bad smell.
- The structure or the property from where the proposed smell originates.
- The possible post-stage impact of the appearance of the proposed smell.
- The possible fixes to the detection of the instances of the proposed bad smell.
- The proposed bad smells are formally defined in preparation for the detection of their instances.
- An example of an instance of the proposed bad smell (taken from the running example introduced in Fig. 1 with appropriate modifications when needed).

Our research team has extensive experience in working with GRL goal models in both academic and industrial settings. This hands-on experience has been crucial in observing and analyzing common quality issues that arise during the modeling process. We leveraged this knowledge to ensure that our proposed bad smells address practical challenges faced by modelers, thus making our work relevant and valuable to the modeling community.

Creating a high-quality goal model is an iterative process and evolves over time. To this end, our proposed bad smells are intended to serve as useful indicators and checkpoints throughout the entire modeling process rather than being applicable only to completed models. Identifying and addressing these bad smells at earlier stages can contribute to incrementally building a more robust and high-quality model. While some bad smells might temporarily appear in models under development, their presence can still be beneficial for modelers to be aware of during the modeling process. By monitoring and addressing these bad smells as they emerge, modelers can ensure that they do not appear in the final version of the model, which leads to a more refined and better-structured goal model. This proactive approach to model improvement enhances the overall quality of the goal models and reduces the risk of accumulating bad smells in the final model.

4.1 Negligible sub-model

Description: A negligible sub-model is a root goal/softgoal with an importance value of zero.
Origin: It originates when a root goal or softgoal has a value of null or zero as an importance value.
Impact: A root element, which typically breaks down into sub-elements and forms a tree-like structure, signifies the primary goal to be accomplished. When a root element lacks an importance value, it indicates an incomplete specification. However, intermediate elements without an importance value (non-root elements) can still contribute to fulfilling the root element's objectives. As long as the root element has a designated importance, the absence of importance for intermediate elements will not impact the overall satisfaction of an actor. The existence of this smell creates several issues. Including unimportant elements in the model increases its complexity if the subject goal/softgoal and the associated sub-model are not essential to the containing actor or model. In cases where a GRL sub-model is vital to the containing actor or model, setting a zero-importance value or leaving it unset denotes an incomplete specification. Importance values of elements influence satisfaction analysis during goal modeling, and their absence affects the analysis outcomes. Null or zero importance values of elements impact the satisfaction of containing actors and play a crucial role in prioritizing requirements and resolving conflicts when they arise.
Possible fixes: Upon detecting an instance of this smell, the modeler needs to revisit the infected goal/softgoal to evaluate its importance to the containing actor or to the containing model. If it turns out that this goal/softgoal and the associated sub-model are important to the subject actor, the importance value of this element should be changed accordingly. If the containing actor turns to be unimportant to the containing actor or model, this element and the associated sub-model should be removed from the containing actor or model such that fixing this smell does not affect the rest of the model. For example, if an element of that sub-model contributes to other parts of the model, this element might be kept in the model to preserve the completeness of the model.

Definition:
Let *ihg* be the importance of a root goal/softgoal RG in model *GM*
ifihg = 0, the sub-model rooted by RG is deemed as an instance of the negligible sub-model bad smell.

Example: Fig. 2 shows two sub-models in a model. The first is the "Identity managed" and the second is the "Usable system". The "Identity managed" goal has an importance value of zero (**note:** zero importance is not shown). The "Usable system" softgoal has an importance of "H" (i.e., High). Therefore, the "Identity managed" is an example of an instance of the negligible sub-model bad smell.

4.2 Non-strategic sub-model

Description: A non-strategic sub-model is a sub-model that starts the refinement tree with a task or resource.
Origin: Tasks and resources at the root of the refinement tree. The root elements in any sub-model should be goals or softgoals to justify the need for system-oriented requirements. These goals and softgoals delineate the strategic need for the system and its features by linking them to the stakeholders' business and operational objectives.

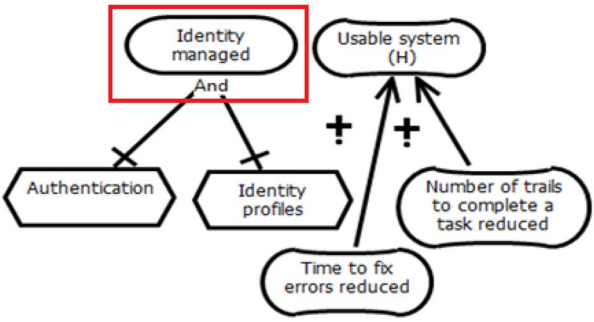

Fig. 2 An example of an instance of negligible sub-model bad smell

Impact: The problem associated with this bad smell is the absence of a justification for the need for the functionality of the infected sub-model. This justification is important as it has several roles. First, it provides a validation tool. It helps validate the selected system-oriented requirements by linking them to the objectives of the stakeholders. It also provides a mechanism to evaluate the different alternatives for achieving the same goal or softgoal.
Possible fixes: Upon detecting an instance of the non-strategic sub-model bad smell, the modeler needs to revisit the model to add the justification of the need of this sub-model by introducing a goal or softgoal as a parent. If this sub-model is turned out to be unjustifiable concerning the real needs of the stakeholders, this sub-model needs to be removed.

Definition: Let *HTR* be a root task or resource in model *GM*,
Let *Parents*(*HTR*) returns the number of parents of *HTR*,
If *Parents*(*HTR*) = 0, *HTR* is deemed as an instance of the non-strategic sub-model bad smell.

Example: Fig. 3 shows an example of an instance of a non-strategic sub-model bad smell. The "Authentication" task is a root task that is not associated with a higher-level goal or softgoal. Therefore, it is deemed as an instance of the non-strategic sub-model bad smell.

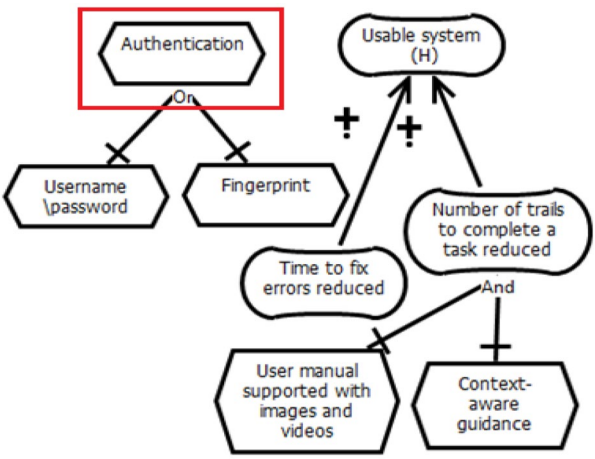

Fig. 3 An example of an instance of a non-strategic sub-model bad smell

4.3 Un-Operationalized Objective

Description: The un-operationalized objective is a leaf goal/softgoal, i.e., left without operationalization. In other words, this goal/softgoal is not refined into tasks or resources.
Origin: Goals or softgoals in the leaves of the refinement tree. These goals or softgoals should not have been left without operationalization.
Impact: The presence of an un-operationalized goal or softgoal leaves the model open to interpretations with respect to the achievability of the infected goal or softgoal. It also makes tracking the achievability of the infected goal or softgoal difficult.
Possible fixes: Upon detecting an instance of the un-operationalized objective bad smell, the modeler needs to revisit the model and make the appropriate modifications. If the un-operationalized objective is found unachievable, this goal or softgoal should be removed from the model. If the un-operationalized objective is found achievable, this goal or softgoal should be operationalized.

Definition: Let O be a goal or softgoal in GRL model GM,
Let $Children(O)$ returns the number of children of O,
If $Children(O) = 0$, O is deemed as an instance of the un-operationalized objective bad smell.

Example: Fig. 4 shows an example of an instance of the un-operationalized objective bad smell. The "Time to fix errors reduced" softgoal is a leaf element that represents an instance of the un-operationalized bad smell (i.e., it is not refined to tasks or resources). Contrary to the "Time to fix errors reduced" softgoal, the "Number of trails to complete a task reduced" softgoal is operationalized; hence it does not exhibit this bad smell.

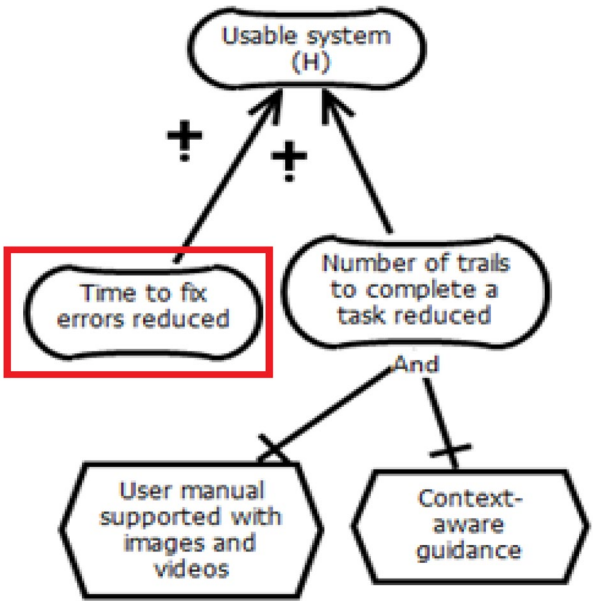

Fig. 4 An example of an instance of un-operationalized objective bad smell

4.4 Disturbed operationalization

Description: Disturbed operationalization is a task or resource that has a goal or softgoal as a child.
Origin: The instances of this bad smell originate in the refinement of tasks or resources. The refinement process starts with listing the high-level goals and softgoals. Then, as needed, these goals and softgoals are refined into other goals and softgoals. Once the objectives of the stakeholders are clearly defined, the modeler needs to define mechanisms to achieve these objectives. Therefore, the modeler needs to refine the lowest-level objectives into tasks and resources.
Impact: Disturbed operationalization might be a sign of detailed modeling, especially when goals or softgoals are used to describe a condition or a property to be held by the respective task or resource. Detailed modeling is a bad symptom in goal models because it draws the attention of the modeler from the high-level objectives to low-level details losing the benefits promised by goal modeling. If disturbing goals or softgoals are conditions or properties to the disturbed task or resource, these disturbing goals or softgoals are details that should be left to later stages. Furthermore, the disturbing goals and softgoals create confusion for the consumers of the model. Understanding the model tends to be more difficult as the readers will be trying to relate those disturbing goals and softgoals into the domain while they are describing aspects of the system to be developed.
Possible fixes: Upon detecting an instance of the disturbed operationalization bad smell, the modeler can apply a fix depending on the context of the detected instance. If the content of the disturbing goal or softgoal represents a mechanism for operationalization, the type of the disturbing element needs to be changed into the appropriate type (i.e., a task or resource). If the content of the disturbing goal or softgoal represents details, this disturbing element and the associated lower-level elements need to be removed.

Definition: Let DO be a task or a resource in GRL model GM,
Let $C = \{c1, c2, \ldots, cn\}$ be the set of the children of DO,
Let $TypesOf(C)$ returns the set of types of the children of DO (i.e., goal, softgoal, task, or resource),
If $TypesOf(C) \cap \{goal, softgoal\}$, DO is deemed as an instance of the disturbed operationalization bad smell.

Example: Fig. 5 shows an example of an instance of the disturbed operationalization bad smell. The "Authentication" task is disturbed by the "Be encrypted" goal. This task is deemed as an instance of the disturbed operationalization bad smell.

4.5 Unknown contribution

Description: Unknown contribution is a contribution link with Unknown or zero contribution.
Origin: Zero (i.e., quantitative) or Unknown (i.e., qualitative) values leave the model open to interpretations in terms of the nature of the relationship (i.e., positive or negative) and in terms of the extent of the relationship (i.e., strong or weak).
Impact: The presence of this smell indicates an incompleteness in the requirements specification. It leaves the infected link open to interpretations. It is difficult to tell whether it contributes positively or negatively. Also, it is difficult to tell the extent of this contribution. Therefore, several post-stage analysis endeavors might be hindered.

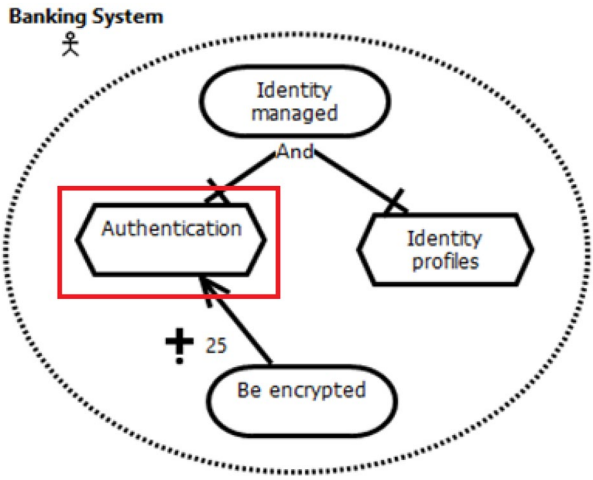

Fig. 5 An example of an instance of disturbed operationalization bad smell

For example, if the respective consumers of the created model know that an element is contributing negatively to a high-level goal, they might take measures to evaluate and react to this contribution.
Possible fixes: upon detecting an instance of this smell, the modeler needs to revisit the model and change this value to a representative value based on the gathered requirements and domain information.

Definition: Let *CL* be a contribution link in GRL model *GM*,
Let *cl* be the contribution value associated with link *CL*,

$$ifcl = 0 orcl = unknown,$$

CL is deemed as an instance of the unknown contribution bad smell.

Example: Fig. 6 shows an example of an instance of the unknown contribution bad smell. The "Time to fix errors reduced" softgoal has an unknown contribution to the "usable system" softgoal. This contribution is deemed as an instance of the unknown contribution bad smell. The situation is different for the "Number of trails to complete a task reduced" softgoal. This softgoal has a contribution value of "+" (i.e., Help).

5 Evaluation of the proposed bad smells

To evaluate the proposed bad smells, a questionnaire was developed and sent to experts.[3] The objective of this questionnaire is to survey experts' opinions on the proposed bad smells.

[3] https://docs.google.com/forms/d/e/1FAIpQLSd76XqT-7QHwG8ctCocbB6kVRbHqCF-pjyEyf52NBhxS48V_w/viewform

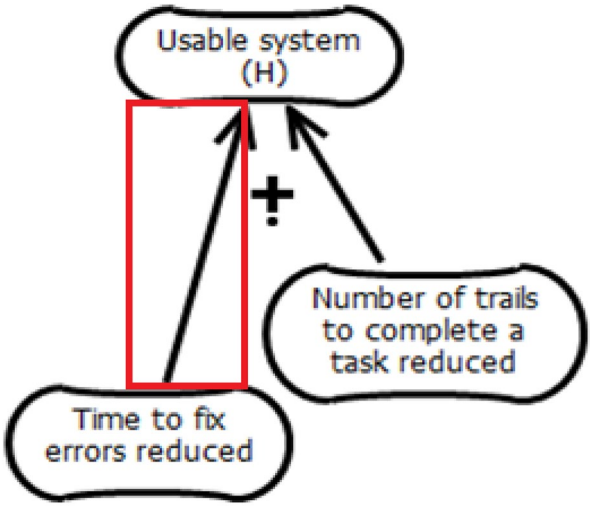

Fig. 6 An example of an instance of unknown contribution bad smell

5.1 Questionnaire design

This questionnaire is structured into three parts. The first part is designed to assess the participants' background in knowledge areas related to the objective of this questionnaire, namely, requirements engineering, goal modeling, GRL, and bad smells. A qualitative scale of 5 categories is used to describe their experience with these areas as follows:

1. Expert (recognized authority)
2. Advanced (applied theory)
3. Intermediate (practical application)
4. Novice (limited experience)
5. Fundamental Awareness (Basic knowledge)

The second part is the core part. It consists of several sections. Each section is designed to evaluate one of the proposed bad smells and includes a description of the subject bad smell, its possible impacts, and a visual example. A qualitative scale of 5 categories is used to describe the opinions of experts with the articulated symptom as a bad smell as follows:

1. Strongly Agree
2. Agree
3. Neutral
4. Disagree
5. Strongly Disagree

Finally, the third part is designed to collect the participant's contact information (if willing to share) and any suggestions or comments on the survey in general.

Table 2 Background of the participant

Level	Experience with Requirements engineering	Experience with goal-oriented modeling	Experience with GRL	Experience with bad smells
Expert	6	3	3	1
Advanced	5	7	6	5
Intermediate	4	3	3	4
Novice	1	2	2	3
Awareness	2	3	4	5

5.2 Results synthesis mechanism

When consolidating the findings, we employed majority voting as an appropriate method for addressing the subjective nature of bad smells, which may be perceived as problematic by some individuals but not by others. Consequently, a complete consensus among respondents regarding the categorization of assessed symptoms as bad smells might not be achievable. Modelers and analysts are likely to have varying opinions on these symptoms based on factors such as their personal assessments, the type of project, and the system domain.

5.3 Participants background

We forwarded this questionnaire to more than 50 possible participants, including requirements researchers and engineers. As a result, 18 participants responded to this questionnaire. These participants were found to have different levels of experience related to the subject of this questionnaire, as shown in Table 2.

5.4 Survey results and discussion

Figure 7 presents a summary of expert opinions, which is divided into two sections. The first section displays the evaluated bad smells, while the second section indicates the number of votes assigned to each respective bad smell. These votes are categorized into three groups: agreement (agree and strongly agree), neutral, and disagreement (disagree and strongly disagree). It is evident that the majority of participants concur that

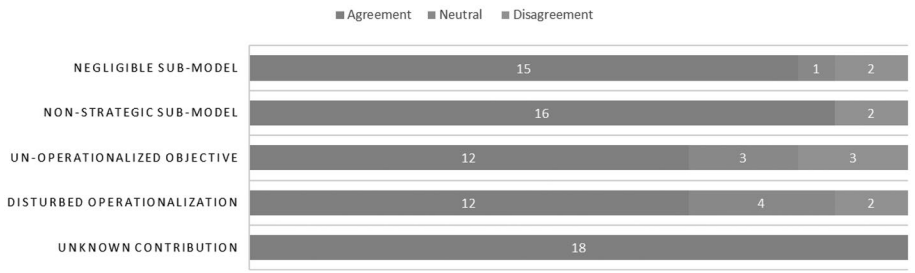

Fig. 7 Bad Smell Articulation Survey Results

the described symptoms are indicative of bad smells. The assessment of the identified bad smells indicates a significant consensus, with the majority of evaluations aligning closely: 83.3% agreement for smell 1, 88.8% for smell 2, and 66.6% for both smells 3 and 4, while all assessments unanimously agree on smell 5.

We further analyzed the comments provided by the respondents to extract further insights on the qualitative feedback for each bad smell.

For the negligible sub-model bad smells, respondents highlighted the impact of not specifying importance values in goal models. One respondent expressed concerns about the undefined importance values assigned to certain elements within the goal model. He argued that the absence of importance can result in inconsistent prioritization and potential difficulty in decision-making. Another respondent emphasized that the absence of assigned importance does not necessarily imply insignificance, potentially resulting in ambiguity regarding prioritization. A third respondent proposed the idea of maintaining consistency in the specification of importance across the model, particularly advocating for its application to non-root elements as well. He stated that this would ensure uniformity and clarity throughout the model. Overall, there is a consensus among respondents that specifying importance values is essential for avoiding confusion and ensuring proper prioritization.

The non-strategic sub-model was identified as a clear issue, as the absence of clear goals or softgoals makes it challenging to prioritize requirements and make informed decisions regarding their significance. This deficiency may result in a system that inadequately addresses stakeholder needs and objectives. Moreover, the lack of traceable justification for tasks impedes analysis and validation efforts. Establishing links between tasks and goals/softgoals is essential to provide necessary rationale, although concerns were raised regarding potential misalignment between tasks and associated goals.

Respondents emphasized that the presence of an Un-operationalized Objective bad smell may lead to confusion, ambiguity, and unwarranted assumptions among designers and coders who will implement the system, resulting in wasted effort. Furthermore, the lack of operationalization makes it difficult to track the progress and effectiveness of these objectives, hindering both prioritization and resource allocation efforts and potentially jeopardizing project success. However, in the early phases, it might not be clear how to satisfy the goal, but it is important to have it there until a complete view of the system is achieved.

Several respondents concurred that the distributed operationalization smell indicates a lack of separation between strategic goals and implementation-specific details, potentially impeding the achievement of overarching objectives and goal modeling effectiveness.

Respondents unanimously agreed on the importance of defining contribution values, even if qualitatively, to address the Unknown Contribution smell. Failing to do so introduces challenges in assessing and analyzing satisfaction levels, impeding decision-making.

6 The proposed detection approach

The instances of the identified GRL bad smells, introduced in Sect. 4, have to be detected as the first step to rectify them. However, detecting these symptoms manually is a cumbersome task due to the size, complexity, and nature of GRL models. Therefore, an automatic detection technique is needed to detect instances of the proposed bad smells. To this end, the structural properties of GRL models and the nature of the proposed bad smells are investigated. Accordingly, a rule-based technique using Object Constraints Language

(OCL) (Booch et al., 1997) rules are used in this work to detect the instances of the proposed bad smells. OCL is an expressive and precise rules development language (Cabot & Gogolla, 2012). It is used to develop efficient and concise queries. It has been used to detect instances of code smells (Kim & Kim, 2008; Kim et al., 2013) and instances of model bad smells such as use-case models (El-Attar & Miller, 2010) and class models (Enckevort, 2009; Štolc & Polášek, 2010). Besides, using OCL rules allows for integrating the proposed bad smells with the previously existing rules in jUCMNav tool (Yan, 2008) (see Sect. 7.2).

6.1 Problem formulation

In this section, the problem of detecting the proposed bad smells is formulated as a constraint checking problem. According to this representation, the proposed bad smells are represented as violations of constraints. In other words, constraints will be used to augment the GRL meta-model, and these constraints are expected to be preserved by input GRL models. Otherwise, each violation represents an instance of the corresponding bad smells. The constraint checking problem is the problem of verifying whether a given model structure preserves some constraints. The theory behind constraints checking is shown in Fig. 7. Constraints checking starts by taking the input GRL model and the constraints that should be preserved by that model. The evaluation engine applies each constraint against the input model. If a constraint does not hold by the input model, it is reported as a violation of that constraint (i.e., an instance of the corresponding bad smell) Fig. 8.

A deeper look into the developed constraints and the way the evaluation engine uses them is provided in the following sub-sections.

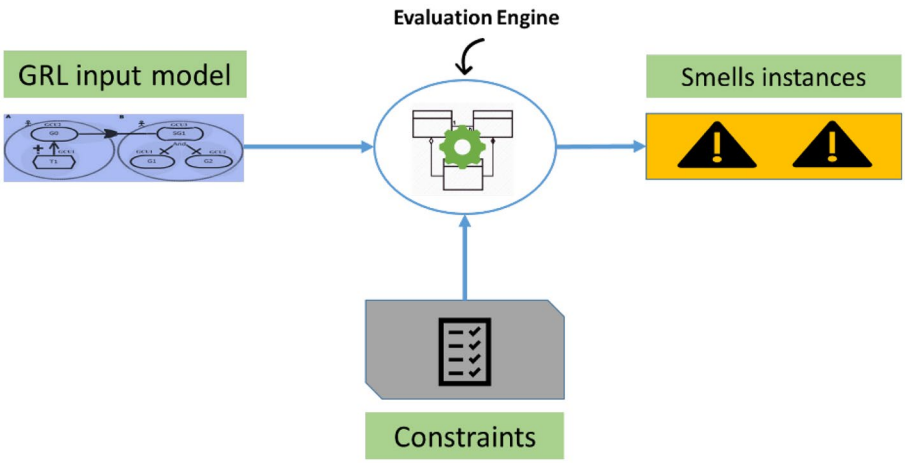

Fig. 8 Properties checking theory and its application for bad smells detection

6.2 Constraints

Constraints represent the core of the developed detection technique. These constraints define conditions that, if preserved by the input model, the input model is deemed free of instances of the respective bad smells. If a constraint is violated, an instance of the corresponding bad smell is reported for each violation. When developing constraints based on OCL, the features (i.e., objects or attributes) to be accessed depends on the adopted target model. The target model can be the user input model or the meta-model of the input model. In this work, the developed constraints are built on top of the GRL meta-model (i.e., meta-model of the input model) (ITU-T, 2018). Input models have limited features. Usually, they do not include all possible features as per the meta-model. Several features are added only when they are set by the modeler. To access all features, the GRL meta-model is used instead.

The core GRL language meta-model (ITU-T, 2018) contains the features needed to build the developed properties that are used to detect the instances of the proposed bad smells. First, the importance of actors and the importance of intentional elements which are inherited from GRLLinkableElement. In GRLLinkableElement, the importance has two representations: quantitative and qualitative. The quantitative importance is named importanceQuantitative and takes a value between 0 and 100. Initially, it is initialized to 0. The qualitative importance is named importance and takes value from the values in the ImportanceType enumeration. Initially, it is initialized to None. Second, the elements' sources and destinations. Sources and destinations help navigate GRL models through ElementLink. ElementLink is associated with the linksSrc and linksDest, which allow accessing the elements' sources and destinations. Third, intentional element types. IntentionalElement contains the type of the attribute. This attribute allows for accessing the type of element. This attribute is initialized with a value from the values in the intenationalElementType enumeration. Fourth, contribution type. In the Contribution class, the contribution has two representations: quantitative and qualitative. The quantitative contribution is named contributionQuantitative and takes an integer value between 0 and 100. Initially, it is initialized to 0. The qualitative contribution is named contribution, and it takes a value from the values in the ContributionType enumeration. Initially, it is initialized to Unknown.

Constraints developed using OCL have three components:

1. Context: represents the scope in which the constraint is applied. It can be a class or an interface. In GRL, it represents actors, elements, links, etc.
2. Characteristics: represent an attribute in the context in which the constraint is applied. In GRL, it represents an attribute of actors, elements, links, etc., such as the importance of an element.
3. Operations and keywords: represent operations that can be applied to qualify or manipulate an attribute."$>$" is an example of an operation. Keywords are terms that are used to specify conditions. For example, if, else, implies, etc. such in, $if\ size() > 0$.

An example of the developed constraints is shown in Fig. 9. The other constraints are provided in Appendix A. In Fig. 9, a constraint is defined to detect the instances of the negligible sub-model bad smell. It provides the evaluation engine with the logic needed to detect the instances of the negligible objective bad smell. The second component defines the query that is used in retrieving the objects that will be searched

Context
$grl::IntentionalElement$
Query
$self.grlspec.intElements \rightarrow select\ (ie \mid not(ie.oclIsTypeOf(grl::kpimodel::Indicator))) \rightarrow asSequence()$
Rule
$(if\ self.grlspec.actors \rightarrow size() = 0\ then\ (not\ (\ (self.linksSrc \rightarrow isEmpty()\ )\ and\ not\ (self.type.toString() = 'Indicator'\ or\ self.type.toString() = 'Task'\ or\ self.type.toString() = 'Ressource'\ )\ and\ self.importanceQuantitative.toString() = '0'))\ else$ $(not\ (\ (self.linksSrc \rightarrow isEmpty()\ )\ and\ not\ (self.type.toString() = 'Indicator'\ or\ self.type.toString() = 'Task'\ or\ self.type.toString() = 'Ressource'\ )\ and\ self.importanceQuantitative.toString() = '0'\ or\ self.refs.contRef.toString().equalsIgnoreCase('null') \rightarrow first())\ endif)$

Fig. 9 The constraint used to detect the instances of the negligible bad smell

for the possible instances of the negligible objective bad smell. The third component defines the rule that checks each of the retrieved objects (i.e., each intentional element) for the instances of the negligible objective bad smell. This rule starts by checking whether it is going to work on the model level or the actor level. If the model contains actors, in each actor, the constraint verifies whether the importance of each high-level goals or softgoals is not zero. If it is zero, meaning that the constraint is not held, the corresponding goal or softgoal is reported as an instance of the negligible objective bad smell.

The functionality of the evaluation engine can be viewed as a three steps process with respect to the defined constraints. In the first step, the engine specifies the context in which the constraint is applied. This context defines the scope of the accessed attributes by the constraint. If the same name is used for naming two attributes in two contexts, the constraint accesses the attribute in the specified context. For example, if "ID" is a name of an attribute in the actor context and in the element context, but the specified context is the element context, the constraint accesses the "ID" attribute in the element context. For accessing the "ID" attribute in the actor context, a fully qualified name starting with the current context should be used. In the second step, the evaluation engine uses the detection query to retrieve all the applicable objects to be checked by the detection rule. In the third step, the evaluation engine uses the detection rule to check each of the retrieved objects. If this constraint is not held on an object, this object is reported as an instance of the specified bad smell.

7 Evaluation of the detection approach

In this section, we present the evaluation of the proposed technique. We start by presenting the evaluation models. Then, we present the developed jUCMNav extension, which implements the developed approach. Finally, we present and discuss the results of running the developed extension on evaluation models.

7.1 Evaluation models

The evaluation models used to evaluate the proposed technique consist of 12 GRL models having different configurations. They fall into three categories from the perspective of the development setting. The first set of models is developed by researchers and published in several papers (Baslyman et al., 2017; Georg et al., 2015; Liu et al., 2009; Neace et al., 2018; Silva et al., 2011; Yu, 2011). The second group of models is created by students, and it is anticipated that these models will be of lower quality. Generally, students lack experience and expertise in goal modeling. Furthermore, they are not subjected to the demands and challenges found in industrial environments. As indicated in Table 3, two of these models were sourced from students' Github accounts, while the remaining three were obtained from projects completed by our students. The third category consists of one industrial model. This model is visually intense and overwhelming with a large number of elements, and its elements are full of text and big in size. In general, this model reflects the characteristics expected in industrial settings.

The evaluation models belong to different domains, including healthcare, office, banking, etc.; moreover, they have different sizes. The details of evaluation models and their sources are shown in Table 3. In terms of actors, we can see that two of these models do not have actors, and one model has only one actor. The rest of the models have more than one actor. In terms of elements and links, the minimum number of elements and links are 12 and 17, respectively, and the maximum number of elements and links are 159 and 232, respectively. The evaluation models used in this work are collected and published as a group online.[4]

7.2 Tool support

An extension is developed to promote the adoption of the proposed smells and technique. This extension is integrated into the jUCMNav Eclipse plugin (version 7.0). Therefore, having the jUCMNav plugin installed is the only requirement to download and use our extension.

To detect instances of the proposed bad smells using our extension, the user needs to follow a three-step procedure. First, open the GRL model required to be analyzed. Second, select bad smells that need to be detected. The user can select all bad smells or some of them, as needed. Third, run the extension and collect the obtained results, as shown in Fig. 10. The results are presented in the problems view of the Eclipse framework. Each row represents an instance of a bad smell. This extension was used to conduct the experiments done in this paper. Information needed to download and use this extension can be found on GitHub.[5]

7.3 Results and discussion

The obtained results are shown in Table 4. Since the developed tool successfully detects all the instances of bad smells as designed, the precision and recall of the developed tool

[4] http://softwareengineeringresearch.net/GRLBSModelCheckingUsingOCLRules.html

[5] https://github.com/MawalMohammed/GRL-bad-smells-detection-based-on-Model-checking

Table 3 Descriptive statistics of the evaluation models

Model #	No. of Actors	No. of Elements	No. of Links	No. of Goals	No. of Softgoals	No. of Tasks	No. of Resources
Models developed by experts for research purposes							
PM1 (Silva et al., 2011)	2	25	32	8	3	10	4
PM2 (Neace et al., 2018)	4	49	90	25	0	13	11
PM3 (Liu et al., 2009)	0	12	18	0	9	3	0
PM4 (Georg et al., 2015)	5	18	20	8	1	6	3
PM5 (Baslyman et al., 2017)	4	14	17	7	0	7	0
PM6 (Yu, 2011)	3	29	39	5	7	15	2
Models developed by students							
SM1[a]	2	27	35	14	0	13	0
SM2[b]	1	25	30	15	0	10	0
SM3	9	54	52	14	15	24	1
SM4	4	159	181	29	33	93	4
SM5	3	19	20	6	3	6	4
Industrial model							
RM[c]	0	151	232	24	67	21	52

[a] https://github.com/mngola/ECSE539A1/blob/3cfc7934de8cc3839ece22d34ccd58e7fc90d047/healthTracker.jucm

[b] https://github.com/McGill-ECSE326-Fall2019-06/Deliverable3/blob/58b524bffd8a1fe44863ed17d2a9f143c0aa0889/Group5SabinaSasu.jucm

[c] https://github.com/ascetictoolbox/ascetictoolbox/blob/742b056d7b98f11f5edaf90aa8545410210eefe1/saas/saas-modelling/ascetic-experiment-based-methodology/ASCETIC-Experiment-SaaS-Reqs-Design-Approach.jucm

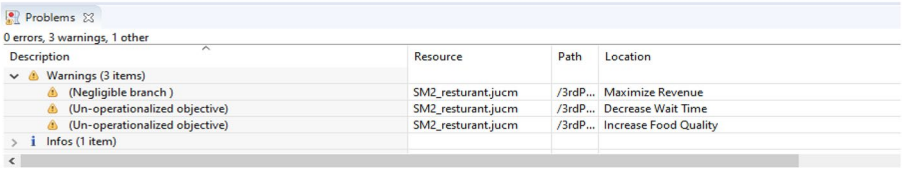

Fig. 10 The output of the tool showing the detected instances of the proposed bad smells

on evaluation models are 100% and 100%, respectively, as the developed technique is deterministic.

In the rest of this section, we present the results and discussion of our analysis. The results are organized by each bad smell as follows:

1. Negligible Sub-model Bad Smell
   Prevalence: Found in all evaluation models except PM3 and PM5.
   Observations:
   - The models sourced from papers exhibit a lower number of instances of this smell. This could potentially be attributed to researchers being more adept at utilizing the GRL framework.
   - RM has the highest number of instances of this smell. The modeler might have considered all goals and softgoals with the same level of importance.
2. Non-strategic Sub-model Bad Smell
   Prevalence: Found in four out of twelve evaluation models (PM6, SM4, PM4, RM).
   Observations:
   - PM6: The appearance of instances of this smell could potentially be attributed to the model being among the earliest ones (Yu, 1997), during which time the semantics had not yet reached maturity or established standardization.
   - SM4: Developed by students who are expected to be less qualified in model development.
   - PM4: In this model, the modeler has operationalized goals and softgoals within other actors.
   - RM: Despite the majority of operationalizations being aligned with the objectives in the model, a number of instances of this bad smell still appear in the model. Some instances can be linked to the model's size and its distribution across multiple graphs, while others may be attributed to the ongoing development of the model.
3. Un-operationalized Objective Bad Smell
   Prevalence: Found in seven out of twelve evaluation models.
   Observations:
   - We found that modelers are aware of the operationalization concept, but some goals and softgoals are left un-operationalized.
   - Scalability problem associated with i* based frameworks might contribute to the appearance of this smell.
   - RM has the largest number of instances, partially due to its size and being under development.

Table 4 the results of the evaluation of the proposed technique

Bad Smells	PM1	PM2	PM3	PM4	PM5	PM6	SM1	SM2	SM3	SM4	SM5	RM
Negligible objective	2	2	0	3	0	1	8	1	9	5	6	21
Non-strategic sub-model	0	0	0	5	0	3	0	0	0	1	0	16
Un-operationalized objective	3	2	2	0	0	0	1	2	12	0	0	28
Disturbed Operationalization	2	0	0	0	0	4	0	0	5	0	0	2
Unknown Contribution	0	0	5	0	0	0	0	0	0	0	0	2

4. Disturbed Operationalization Bad Smell
   Prevalence: Found in four out of twelve evaluation models (PM1, PM6, SM3, RM).
   Observations:
   - We observed that numerous instances of this bad smell are associated with goals used to represent conditions and constraints for tasks. Ideally, these aspects should be addressed later in the requirements engineering process.
   - PM6: Being one of the earliest models, at the time, the semantics were not mature or standardized, leading to task elements being refined to goals and softgoals.
   - SM3: Developed by students, the repetition of disturbed tasks was to delegate goals to different actors.
   - RM: Only two instances were found; size and visual intensity (i.e., the model is crowded with elements that are distributed over several graphs) of the model may have contributed to the appearance of these instances.
5. Unknown Contribution Bad Smell
   Prevalence: Found in two models (PM3, RM).
   Observations:
   - PM3: The presence of smells in this model may be due to its development for a specific purpose within the paper, with minimal focus on other aspects.
   - RM: The size of this model is the first thing to blame for the appearance of the instances of this bad smell.

8 Threats to validity

This work is subject to several threats to validity. These threats can be categorized according to four important categories as identified by Wohlin et al. (Wohlin et al., 2012).

Construct threats: One possible threat is related to the selection of the evaluation models. However, we selected the evaluation models to represent the different domains, sizes, and development environments (i.e., academic and industrial). We are not aware of the circumstances of creating these models and the level of experience of the developers. However, we do not think this affects the effectiveness of the developed tool and technique, as the models used in the evaluation cover various configurations. The developed tool is an open-source tool, and it is available online (see Sect. 7.2) for researchers who want to evaluate this approach under different circumstances.

Internal threats: The internal threats to validity concern the degree to which the outcomes of the study depend on the experimental variables. Based on the obtained results, the proposed detection technique was able to detect all the instances of the proposed bad smells as this approach is deterministic and exact. Hence, it is expected to detect all the instances of the proposed bad smells. If there are instances that have been missed, it will be due to the implementation of the tool. To mitigate that, we tested the tool and manually evaluated the obtained results.

External threats: External threats to validity concern the ability to generalize the outcomes of the study. The proposed bad smells and detection techniques are introduced to address bad smells in goal models developed using GRL. For goal models developed using other goal modeling frameworks such as i* or AGORA, slight modification might be required to be applied for such languages.

Conclusion threats: Conclusion threats to validity concern the threats associated with the conclusions made in this work. The objective of this work was to propose an approach to help modelers identify quality improvement opportunities in GRL models. To this end, a catalog of five bad smells is developed. The instances of these smells are expected to provide opportunities to improve the quality of the addressed GRL goal model. However, as bad smells, the different modelers might interpret these instances differently in different circumstances as shown in the survey results discussion. Therefore, we provided the tool with the option of selecting and deselecting the detection of these smells.

9 Conclusion

GRL models are used in the early stage of the requirements engineering process to identify stakeholders, their needs, and how these needs can be satisfied. Problems in this stage are more difficult and expensive to rectify compared to problems in the later stages in the process of software development. Therefore, generating high-quality GRL models improves software project success rate, reduces development costs, and results in higher-quality software.

In pursuit of higher-quality GRL models, the notion of bad smells is used in this work to pinpoint quality improvement opportunities. To this end, a list of bad smells is proposed and articulated. To evaluate the proposed bad smells, a questionnaire is developed and sent to experts to survey their opinions on the articulation of the proposed bad smell. The experts provided their feedback by assigning votes to each bad smell, which were then categorized into agreement (agree and strongly agree), neutral, and disagreement (disagree and strongly disagree). The results show that most participants concur that the described symptoms indicate bad smells, with only a small number of votes expressing disagreement or neutrality.

A rule-based detection technique is developed using OCL rules to facilitate the detection of instances of these smells. The proposed technique is developed as an extension to jUCMNav to promote its adoption in practice and research settings. The developed technique is then evaluated on a set of evaluation models that are collected from various settings (i.e., an industrial model, models from papers, and models developed by students). The obtained results show that the developed techniques successfully detect all the instances of the developed technique. It achieved a recall and precision of 100%.

In conclusion, based on the obtained results, we believe that the developed technique will effectively help improve the quality of GRL models by suggesting quality improvement opportunities. We also believe that the proposed set of bad smells will indirectly help improve the quality of GRL models by educating modelers on bad-quality symptoms.

In future work, we plan to extend this list of bad smells with additional bad smells. Accordingly, we plan to develop additional bad smell detection techniques if needed. Other future work venues include adapting the proposed bad smells and detection technique to work with the other goal modeling frameworks such as i*, NFR, etc. In addition, we plan to work on refining our approach and establishing a taxonomy for bad smells in future iterations of our work. We believe this will contribute to a more robust and systematic process for identifying and analyzing bad smells in models, ultimately improving the overall quality of model development practices. Moreover,

we plan to explore integrating machine learning, natural language processing, and large language models (such as ChatGPT) into our approach for detecting bad smells and enhancing the refinement process in GRL model development in future iterations of our work.

Appendix A – OCL constraints for the proposed bad smells

Non-Strategic Sub-Model

Context
$grl::IntentionalElement$
Query
$self.grlspec.intElements{->}\ select\ (ie \mid not(ie.oclIsTypeOf(grl::kpimodel::Indicator))\ ){->}\ asSequence()$
Rule
$not\ (\ (self.linksSrc{->}\ isEmpty()\ or\ self.linksSrc{->}\ select(ls\|ls.oclIsTypeOf(grl::Dependency)){->}\ size()\ = self.linksSrc{->}\ size()\ )\ and\ (self.type.toString()\ = \ 'Task'\ or\ self.type.toString()\ = \ 'Ressource'\ ))\ or\ self.refs.contRef.toString().equalsIgnoreCase('null'){->}\ first()$

Un-operationalized Objectives

Context
$grl::IntentionalElement$
Query
$self.grlspec.intElements{->}\ select\ (ie \mid not(ie.oclIsTypeOf(grl::kpimodel::Indicator))\ ){->}\ asSequence()$
Rule
$(self.type.toString()\ = \ 'Indicator'\ or\ self.type.toString()\ = \ 'Task'\ or\ self.type.toString()\ = \ 'Ressource'\ )\ or\ self.linksDest{->}\ size()\ <>\ 0$ $(not\ (\ (self.linksSrc \rightarrow isEmpty()\ )\ and\ not\ (self.type.toString()\ = \ 'Indicator'\ or\ self.type.toString()\ = \ 'Task'\ or\ self.type.toString()\ = \ 'Ressource'\ )\ and\ self.importanceQuantitative.toString()\ = '0'\ )\ or\ self.refs.contRef.toString().equalsIgnoreCase('null') \rightarrow\ first())\ endif)$

Disturbed Operationalization

Context
$grl::IntentionalElement$
Query
$self.grlspec.intElements\ {->}\ select(ie\|ie.type = grl::IntentionalElementType::Task)\ {->}\ asSequence()$
Rule
$self.linksDest{->}\ notEmpty()\ implies\ self.linksDest{->}\ forAll\ (ld\|\ (ld.src.oclAsType(grl::IntentionalElement).type.toString()\ = 'Task'\ or\ ld.src.oclAsType(grl::IntentionalElement).type.toString()\ = 'Ressource')\ or\ ld.oclIsTypeOf(grl::Dependency)\ )$

Unknown Contribution

Context
$grl::Contribution$
Query
$self.grlspec.links\ {->}\ select(link:grl::ElementLink \mid link.oclIsTypeOf(grl::Contribution)){->}\ collect(l\|l.oclAsType(grl::Contribution)){->}\ asSequence()$
Rule
$(self.contribution.toString()\ <>\ 'Unknown'$

Acknowledgements The authors acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

Author contributions Mawal Ali Mohammed wrote the main manuscript text. Mohammad Alshayeb supervised the work and contributed to the manuscript writing. Jameleddine Hassine supervised the work and contributed to the manuscript writing. All authors reviewed the manuscript.

Funding The authors did not receive support from any organization for the submitted work.

Data availability Data used in this work is available at: https://github.com/MawalMohammed/GRL-bad-smells-detection-based-on-Model-checking

Declarations

Competing interests The authors declare no competing interests.

Financial interests The authors have no relevant financial or non-financial interests to disclose.

References

AbuHassan, A., Alshayeb, M., & Ghouti, L. (2021). Software smell detection techniques: A systematic literature review. *Journal of Software: Evolution and Process, 33*(3), e2320. https://doi.org/10.1002/smr.2320

Alkharabsheh, K., Crespo, Y., Manso, E., & Taboada, J. (2019). Software Design Smell Detection: A systematic mapping study. *Software Quality Journal, 27*(3), 1069–1148.

Amyot, D., Horkoff, J., Gross, D., & Mussbacher, G. (2009). *A lightweight GRL profile for i* modeling.* Paper presented at the Advances in Conceptual Modeling-Challenging Perspectives: ER 2009 Workshops CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS, Gramado, Brazil, November 9–12, 2009. Proceedings 28.

Amyot, D., Mussbacher, G., Ghanavati, S., & Kealey, J. (2011). GRL Modeling and Analysis with jUCMNav. *iStar, 766*, 160–162.

Amyot, D., Akhigbe, O., Baslyman, M., Ghanavati, S., Ghasemi, M., Hassine, J., . . . Yu, E. (2022). Combining Goal modelling with Business Process modelling: Two Decades of Experience with the User Requirements Notation Standard. *Enterprise Modelling and Information Systems Architectures, 17*(2), 1–38.

Arendt, T., & Taentzer, G. J. U. M. (2010). UML model smells and model refactorings in early software development phases. *Universitat Marburg*.

Asano, K., Hayashi, S., & Saeki, M. (2017). *Detecting Bad Smells of Refinement in Goal-Oriented Requirements Analysis.* Paper presented at the International Conference on Conceptual Modeling.

Baqais, A., & Alshayeb, M. (2020). Automatic Software Refactoring: A Systematic Literature Review. *Software Quality Journal, 2*(28), 459–502.

Baslyman, M., Almoaber, B., Amyot, D., & Bouattane, E. M. (2017). *Activity-based process integration in healthcare with the user requirements notation.* Paper presented at the International Conference on E-Technologies.

Bertran, I. M. (2011). *Detecting architecturally-relevant code smells in evolving software systems.* Paper presented at the 2011 33rd International Conference on Software Engineering (ICSE).

Booch, G., Jacobson, I., & Rumbaugh, J. (1997). Object constraint language specification. *UML Documentation Set Version, 1*.

Boussaa, M., Kessentini, W., Kessentini, M., Bechikh, S., & Chikha, S. B. (2013). *Competitive coevolutionary code-smells detection.* Paper presented at the International Symposium on Search Based Software Engineering.

Cabot, J., & Gogolla, M. (2012). *Object constraint language (OCL): a definitive guide.* Paper presented at the International school on formal methods for the design of computer, communication and software systems.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5): Springer Science & Business Media.

Czibula, G., Marian, Z., & Czibula, I. G. (2015). Detecting software design defects using relational association rule mining. *Knowledge and Information Systems, 42*(3), 545–577.

Dardenne, A., van Lamsweerde, A., & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming, 20*(1–2), 3–50. https://doi.org/10.1016/0167-6423(93)90021-g

DeMarco, T. (1979). Structure analysis and system specification. *Pioneers and Their Contributions to Software Engineering* (pp. 255–288). Springer.

Denger, C., & Olsson, T. (2005). Quality assurance in requirements engineering. *Engineering and managing software requirements* (pp. 163–185). Springer, Berlin: Heidelberg.

Dexun, J., Peijun, M., Xiaohong, S., & Tiantian, W. (2012). *Detecting bad smells with weight based distance metrics theory.* Paper presented at the Second International Conference on Instrumentation, Measurement, Computer, Communication and Control.

El-Attar, M., & Miller, J. (2010). Improving the quality of use case models using antipatterns. *Software and Systems Modeling, 9*(2), 141–160.

Enckevort, T. v. (2009). *Refactoring UML models: using openarchitectureware to measure uml model quality and perform pattern matching on UML models with OCL queries.* Paper presented at the Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.

Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., & Zimmer, J. (2014). *Rapid requirements checks with requirements smells: two case studies.* Paper presented at the Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering.

Femmer, H., Hauptmann, B., Eder, S., & Moser, D. (2016). *Quality assurance of requirements artifacts in practice: A case study and a process proposal.* Paper presented at the International Conference on Product-Focused Software Process Improvement.

Femmer, H. (2017). *Requirements engineering artifact quality: definition and control.* Technische Universität München,

Fontana, F. A., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., & Di Nitto, E. (2017). *Arcan: a tool for architectural smells detection.* Paper presented at the IEEE International Conference on Software Architecture Workshops (ICSAW).

Fontana, F. A., Mäntylä, M. V., Zanoni, M., & Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering, 21*(3), 1143–1191.

Fowler, M. (2018). *Refactoring: improving the design of existing code*: Addison-Wesley Professional.

Georg, G., Mussbacher, G., Amyot, D., Petriu, D., Troup, L., Lozano-Fuentes, S., . . . Technology, S. (2015). Synergy between Activity Theory and goal/scenario modeling for requirements elicitation, analysis, and evolution. *59*, 109–135.

Ghanavati, S., Amyot, D., & Rifaut, A. (2014). *Legal goal-oriented requirement language (legal GRL) for modeling regulations*. Paper presented at the Proceedings of the 6th International Workshop on Modeling in Software Engineering, Hyderabad, India. https://doi.org/10.1145/2593770.2593780

Hofmann, H. F., & Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE Software, 18*(4), 58–66.

Horkoff, Aydemir, Cardoso, E., Li, T., Maté, A., Paja, E., . . . Giorgini, P. (2016, 12–16 Sept. 2016). *Goal-Oriented Requirements Engineering: A Systematic Literature Map.* Paper presented at the 2016 IEEE 24th International Requirements Engineering Conference (RE).

Horkoff, J., & Yu, E. (2013). Comparison and evaluation of goal-oriented satisfaction analysis techniques. *Requirements Engineering, 18*(3), 199–222.

Hozano, M., Antunes, N., Fonseca, B., & Costa, E. (2017). *Evaluating the Accuracy of Machine Learning Algorithms on Detecting Code Smells for Different Developers.* Paper presented at the ICEIS (2).

ITU-T, Z. (2018). 151 User requirements notation (URN)–Language definition. *ITU-T*. Retrieved from https://www.itu.int/rec/T-REC-Z.151/en

Kaiya, H., Horai, H., & Saeki, M. (2002). *AGORA: Attributed goal-oriented requirements analysis method.* Paper presented at the Proceedings IEEE joint international conference on requirements engineering.

Katasonov, A., & Sakkinen, M. (2006). Requirements quality control: A unifying framework. *Requirements Engineering, 11*(1), 42–57.

Kessentini, M., Sahraoui, H., Boukadoum, M., & Wimmer, M. (2011). *Design Defect Detection Rules Generation: A Music Metaphor.* Paper presented at the 15th European Conference on Software Maintenance and Reengineering.

Kessentini, W., Kessentini, M., Sahraoui, H., Bechikh, S., & Ouni, A. (2014). A cooperative parallel search-based software engineering approach for code-smells detection. *IEEE Transactions on Software Engineering, 40*(9), 841–861.

Kim, T.-W., & Kim, T.-G. (2008). Automated code smell detection and refactoring using OCL. *The KIPS Transactions: Partd, 15*(6), 825–840.

Kim, T.-W., Kim, T.-G., & Seu, J.-H. (2013). Specification and automated detection of code smells using ocl. *International Journal of Software Engineering and Its Applications, 7*(4), 35–44.

Knauss, E., El Boustani, C., & Flohr, T. (2009). *Investigating the impact of software requirements specification quality on project success.* Paper presented at the International Conference on Product-Focused Software Process Improvement.

Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering, 160*(1), 3–24.

Lapouchnian, A. J. U. o. T. (2005). Goal-oriented requirements engineering: An overview of the current research. *32*.

Lee, K. S., & Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. *Computer Methods in Applied Mechanics Engineering, 194*(36–38), 3902–3933.

Liu, X., Peyton, L., & Kuziemsky, C. (2009). *A requirement engineering framework for electronic data sharing of health care data between organizations.* Paper presented at the International Conference on E-Technologies.

Lones, M. (2011). Sean Luke: essentials of metaheuristics. In: Springer.

Maneerat, N., & Muenchaisri, P. (2011). *Bad-smell prediction from software design model using machine learning techniques.* Paper presented at the Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE).

Misbhauddin, M., & Alshayeb, M. (2015). UML model refactoring: A systematic literature review. *Empirical Software Engineering, 20*(1), 206–251.

Mohammed, M. A., Alshayeb, M., & Hassine, J. (2022a). A search-based approach for detecting circular dependency bad smell in goal-oriented models. *Software and Systems Modeling, 21*(5), 2007–2037. https://doi.org/10.1007/s10270-021-00965-z

Mohammed, M. A., Hassine, J., & Alshayeb, M. (2022b). GSDetector: A tool for automatic detection of bad smells in GRL goal models. *International Journal on Software Tools for Technology Transfer, 24*(6), 889–910. https://doi.org/10.1007/s10009-022-00662-2

Mumtaz, H., Alshayeb, M., Mahmood, S., & Niazi, M. (2019). A survey on UML model smells detection techniques for software refactoring. *Journal of Software: Evolution and Process, 31*(3), e2154.

Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering, 18*(6), 483–497.

Neace, K., Roncace, R., & Fomin, P. (2018). Goal model analysis of autonomy requirements for Unmanned Aircraft Systems. *Requirements Engineering, 23*(4), 509–555.

Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M., & Nguyen, T. N. (2009). *Graph-based mining of multiple object usage patterns.* Paper presented at the Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering.

Nongpong, K. (2015). *Feature envy factor: A metric for automatic feature envy detection.* Paper presented at the 7th International Conference on Knowledge and Smart Technology (KST).

Ouni, A., Kessentini, M., Inoue, K., & Cinnéide, M. O. (2017). Search-based web service antipatterns detection. *IEEE Transactions on Services Computing, 10*(4), 603–617.

Peiris, M., & Hill, J. H. (2016). *Automatically Detecting Excessive Dynamic Memory Allocations Software Performance Anti-Pattern.* Paper presented at the Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering.

Pohl, K. (2010). *Requirements engineering: fundamentals, principles, and techniques*: Springer Publishing Company, Incorporated.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. E. (1991). *Object-oriented modeling and design* (Vol. 199): Prentice-hall Englewood Cliffs, NJ.

Salger, F. (2013). *Requirements reviews revisited: Residual challenges and open research questions.* Paper presented at the 21st IEEE International Requirements Engineering Conference (RE).

Seki, Y., Hayashi, S., & Saeki, M. (2019). *Detecting Bad Smells in Use Case Descriptions.* Paper presented at the 2019 IEEE 27th International Requirements Engineering Conference (RE).

Sharma, T., & Spinellis, D. (2018). A survey on software smells. *Journal of Systems and Software, 138*, 158–173.

Silva, C. T., Borba, C., & Castro, J. (2011). *A Goal Oriented Approach to Identify and Configure Feature Models for Software Product Lines.* Paper presented at the WER.
Standardization, I. O. f. (2018). ISO/IEC/IEEE 29148: 2018–Systems and software engineering—Life cycle processes—Requirements engineering. In: ISO ISO/IEC/IEEE Switzerland.
Štolc, M., & Polášek, I. (2010). *A visual based framework for the model refactoring techniques.* Paper presented at the 2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI).
Tekin, U., & Buzluca, F. (2014). A graph mining approach for detecting identical design structures in object-oriented design models. *Science of Computer Programming, 95*, 406–425.
Tukur, M., Umar, S., & Hassine, J. (2021). Requirement engineering challenges: A systematic mapping study on the academic and the industrial perspective. *Arabian Journal for Science Engineering, 46*, 3723–3748.
Van Lamsweerde, A. (2000). *Requirements engineering in the year 00: a research perspective.* Paper presented at the Proceedings of the 22nd international conference on Software engineering.
Van Lamsweerde, A. (2001). *Goal-oriented requirements engineering: A guided tour.* Paper presented at the Proceedings fifth ieee international symposium on requirements engineering.
Van Lamsweerde, A., & Letier, E. (2004). From object orientation to goal orientation: A paradigm shift for requirements engineering. *Radical Innovations of Software and Systems Engineering in the Future* (pp. 325–340). Springer.
West, D. B. (1996). *Introduction to graph theory* (5th ed.). Prentice hall Upper Saddle River, NJ.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.
Yan, J. B. (2008). Static Semantics Checking Tool for jUCMNav. In: Master's project, SITE, University of Ottawa.
Yu, E. (1997). *Towards modelling and reasoning support for early-phase requirements engineering.* Paper presented at the Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on.
Yu, E., & Mylopoulos, J. (1998). *Why goal-oriented requirements engineering.* Paper presented at the Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality.
Yu, E. (2011). Modeling Strategic Relationships for Process Reengineering. *Social Modeling for Requirements Engineering* (11th ed., pp. 66–87)
Yu, E., Amyot, D., Mussbacher, G., Franch, X., & Castro, J. (2013). *Practical applications of i* in industry: The state of the art.* Paper presented at the 2013 21st IEEE International Requirements Engineering Conference (RE).
Zelkowitz, M. V., Yeh, R., Hamlet, R. G., Gannon, J. D., & Basili, V. R. (1983). *The Software Industry: A State of the Art Survey*.

Mawal A. Mohammed is an assistant professor at Prince Sattam Bin Abdulaziz University, Saudi Arabia. Dr. Mohammed received his PhD and MSc from the department of Information and Computer science of King Fahd University of Petroleum and Minerals (KFUPM). Prior to this, Dr. Mawal worked as a teaching assistant in the Software Engineering department of Taiz University, Yemen, where he received his BS degree. His research interests include software engineering, requirements engineering, software design patterns, and software quality.

Mohammad Alshayeb is a Professor at the Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia. He received his PhD in Computer Science from the University of Alabama in Huntsville in 2002. Dr. Alshayeb worked as a senior researcher and Software Engineer and managed software projects in the United States and the Middle East. He published over 120 refereed conference and journal publications and holds 5 US patents. He received Khalifa award for education as "the distinguished University Professor in the Field of Teaching within Arab World", 2016. Dr. Alshayeb's research interests include software quality, software measurement, evidence-based software engineering and empirical studies in Software Engineering.

Jameleddine Hassine is an Associate Professor at the department of Information and Computer Science of King Fahd University of Petroleum and Minerals (KFUPM). Dr. Hassine holds a Ph.D. from Concordia University, Canada (2008) and an M.Sc. from the University of Ottawa, Canada (2001). Dr. Hassine has several years of industrial experience within worldwide telecommunication companies; Nortel Networks (Canada) and Cisco Systems (Canada). His main research interests include requirements engineering languages and methods, software testing, formal methods, software evaluation, and maintenance. He is actively involved in several funded research projects and has over 50 publications on various research topics in his field. Dr. Hassine published his research in many high-impact journals like Requirements Engineering Journal (REJ), Journal of Systems and Software (JSS), Information and Software Technology (IST), and Software and Systems Modeling (SoSyM).