2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI) | 979-8-3503-3458-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/IRI58017.2023.00018

Resource-Centric Goal Model Slicing for Detecting Feature Interactions

Zedong Peng, Mahima Dahiya, Tessneem Khalil, Nan Niu
University of Cincinnati
Cincinnati, OH, USA
{pengzd, dahiyama, khaliltm}@mail.uc.edu, nan.niu@uc.edu

Tanmay Bhowmik
Mississippi State University
Mississippi State, MS, USA
tbhowmik@cse.msstate.edu

Yilong Yang
Beihang University
Beijing, China
yilongyang@buaa.edu.cn

***Abstract*—Feature interaction (FI) occurs when the requirements are satisfied by the features in isolation but not in composition. We present a novel approach to FI detection via a lightweight modeling of two features' resource dependency. Our preliminary study on two Zoom features shows three types of resource dependency: produce-and-use, state-changing, and mutual-exclusion. We present the testing pattern associated with each type, report the FI testing results, and discuss our long-term directions toward using real-world software's features to ground and evaluate requirements engineering research.**

***Index Terms*—feature interaction, goal-oriented requirements engineering, requirements-based testing**

I. Introduction

In a software system, a *feature* is an increment of functionality usually with a coherent purpose [1]. A feature is also a client-valued function serving users in a visible way. The concept of a feature is central to the incremental development of complex systems, because it helps to state requirements as largely independent features; in this way, the users can comprehend the features cohesively and the engineers can build them individually [2]. In the modern continuous integration and continuous delivery/deployment (CI/CD) pipeline, software vendors release features to serve their end users, and for that matter, to compete against their rivals as well [3].

Incremental delivery of new and enhanced features means that they must be composed with the existing features to form the whole software system. However, an inherent and significant challenge is the feature interaction (FI) problem, referring to the undesired behavioral modifications that one feature makes to another [4].

FI detection is a complex matter. In theory, the number of possible interactions is $\mathcal{O}(2^{|F|})$ where F is the set of features [5]. In practice, if we focus only on a newly released feature, the number of pairwise interactions to consider is $\mathcal{O}(|F|)$. $\mathcal{O}(|F|)$ is not a trivial bound, e.g., the number of Zoom features released in 2022 is 281 [6]. Without proper support, the development of additional features is ultimately overwhelmed by the FI problem [7].

One such support was introduced by Bisbal and Cheng [8] to address the FI problem on the basis of the *resources* used by features, rather than the features themselves. For instance, if we know the "forward error correction" feature uses $x\%$ battery life and the "image resolution" feature uses $y\%$ battery life, then these two features interact on the basis of the battery as a resource. Despite the prototype implementation presented in [8], Bisbal and Cheng's approach concerned more about quantitatively optimizing a resource. As for FI detection, only a brief mention was made in [8]: "*Intuitively if a given feature has goals that cannot be achieved given the USES and PRODUCES relationships and resources specified for the system, then a feature interaction is detected.*" Here, USES means that a feature is using a resource to perform its task, and PRODUCES means that a feature produces a resource which will be available to this feature or other features [8].

Several practical questions arise: Can a feature's resource(s) be readily recognizable? How exactly do the USES and PRODUCES relationships give rise to FIs? Are there any other resource-based relationships that reveal FIs? Probably most important of all, is an approach centered on resources capable of detecting FIs for a real-world software application like Webex or Zoom used by millions of people? In this paper, we present our ongoing work on constructing resource-centric goal model slices for the purpose of detecting FIs.

Our main contributions are the novel classification of three finder-grained resource dependencies, their testing patterns, and the empirical results obtained over real-world software features. The rest of the paper is organized as follows. Section II lays the foundation of FIs. We then describe our approach and initial results in Section III, discuss related work in Sections IV, and conclude the paper in Section V.

II. What is a Feature Interaction?

Jackson defined the meaning of requirements as "$\mathcal{E}, \mathcal{S} \vdash \mathcal{R}$" in his seminal paper [9], where $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ represent environmental assumptions, specifications, and requirements respectively. The entailment $\vdash$ means that if a software system doing $\mathcal{S}$ is installed in an environment having $\mathcal{E}$, then we know $\mathcal{R}$ will be fulfilled [10]. Peng *et al.* [1] recently viewed a released feature, F, as an implemented $\mathcal{S}$ and thus the feature operated in a given environment will satisfy its requirement, namely, "$\mathcal{E}, F \vdash \mathcal{R}$".

Now, let us consider two different features: "$\mathcal{E}_1, F_1 \vdash \mathcal{R}_1$" and "$\mathcal{E}_2, F_2 \vdash \mathcal{R}_2$". If their composition (denoted as $F_1||F_2$) operating in the shared environment does not satisfy $\mathcal{R}_1$ and $\mathcal{R}_2$ simultaneously, then a FI occurs [2]. More formally, if the requirements are satisfied by the features in isolation:

979-8-3503-3458-6/23/$31.00 ©2023 IEEE
DOI 10.1109/IRI58017.2023.00018

$$\mathcal{E}_1, F_1 \vdash \mathcal{R}_1 \quad (1)$$
$$\mathcal{E}_2, F_2 \vdash \mathcal{R}_2 \quad (2)$$

but not in composition:

$$\mathcal{E}_1 \cap \mathcal{E}_2, F_1 || F_2 \nvdash \mathcal{R}_1 \wedge \mathcal{R}_2 \quad (3)$$

then F_1 and F_2 modify or influence one another in defining overall system behavior, thereby interacting with each other. Although the above characterizations are about pairwise FI, one can easily generalize them to the interaction involving more than two features. From a practical point of view, however, a pairwise FI represents one of the most basic forms that a resolution could help address other interactions. We therefore focus on detecting pairwise FIs in our work.

III. Using Goal Model Slice to Detect Resource-Based Feature Interactions

Our approach advocates the construction of a goal model slice for detecting resource-based FIs. Currently, the goal model slice is built manually by relying on human expertise. While our long-term visions include automating the goal slice modeling, we believe the current manual work is necessary and important to inform automated support development.

We adopt the graphical notations from the i^* framework [11] to depict a goal model slice. Fig. 1-a shows the slice of a single feature, e.g., the "Caller ID Display" feature depends on the "phone screen" (a resource) to help accomplish the goal of "showing who the caller is". When the resource is shared by two features, the goal model slice can be extended to include both features as shown in Fig. 1-b. For instance, the "Caller ID Block" feature depends on the same "phone screen" resource to help fulfill the goal of "hiding who the caller is".

The principle of our goal model slice design is minimality, i.e., choosing only the essential elements for the purpose of uncovering resource-based FIs. Our specific choice of including goals is to explicitly define the oracles [12] for testing individual features; otherwise, no intentional observations would be made to check if the software's actual output or behavior is correct or not. Although i^* does not include feature as a construct, we treat it as a task that carries out the computations in order to fulfill a goal. By considering a minimum set of elements for modeling resource dependency [11], we expect easy constructability because the goal model slice is not meant to be complete or comprehensive.

In order to detect FIs, the goal model slice needs to be as precise and testable as possible. To that end, we refine the resource dependency modeled in Fig. 1-b by presenting our manual analysis results of a Zoom feature. As our analysis was performed in the fall of 2022, we chose a new feature at that time, namely "Decline meeting invitation with message" released in September 2022 [6]. The description of this feature is as follows:

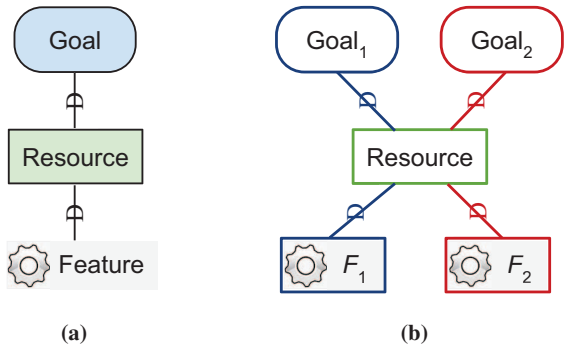


Fig. 1. **(a)** Goal model slice showing that a "Feature" depends on some "Resource" in order to achieve a "Goal", and **(b)** Goal model slice involving two features, F_1 and F_2, which share the same "Resource".

> "When the host of an active meeting invites a contact directly to join, the invitee can send a message when declining the invite, to provide context around the reason for declining. Some preset responses are provided by Zoom, but custom text can be used as well. The message will appear as a chat message in Zoom Team Chat for the person requesting you join."

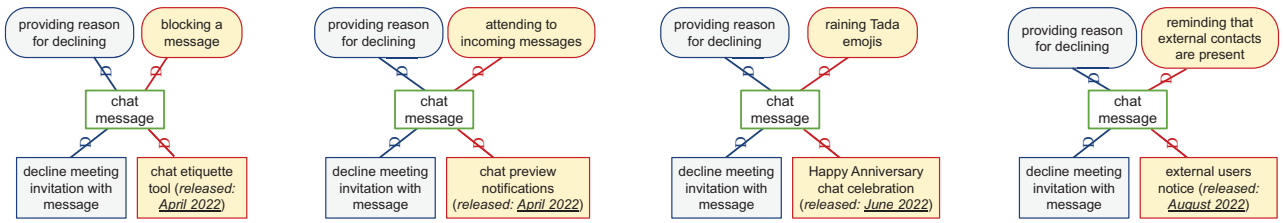
We identified three types of resource dependency: produce-and-use, state-changing, and mutual-exclusion. Each type contained more than one instance, which we present next. For each type, we also contribute the software testing pattern amenable to revealing FIs..

A. Produce-and-Use

Without loss of generality, we refer to the resource-producing feature as feature$_1$, and the resource-using feature as feature$_2$. A familiar scenario of produce-and-use is a text editor's "typing" feature and the "spell checking" feature. Once a resource (e.g., a word) is produced by "typing" (feature$_1$), the same resource will then be used by "spell checking" (feature$_2$) to warn the user in case of misspelling. Regarding Zoom's "Decline meeting invitation with message" feature, it does produce a chat message. Such a produced resource can then be used by several other features, as shown in Fig. 2. Therefore, it is important to test all the goal model slices of Fig. 2.

The testing pattern of resource's produce-and-use FI is shown in Table I. First, feature$_1$ shall be tested in its entirety, without any other feature interfering with it. The test oracle here is to make sure that the resource is indeed produced and the goal of feature$_1$ is satisfied, e.g., a word is actually typed, and a chat message providing reason for declining is in fact composed. This very resource then triggers feature$_2$ (e.g., "spell checking" or "chat preview notifications"), and the expectation (oracle) here is to satisfy the goal of feature$_2$ (e.g., to flag a misspelled word, or to provide more attention to incoming chat messages).

Although the two features' sequencing is linear and straightforward in Table I, they are interdependent. Not only is



(a) **(b)** **(c)** **(d)**

Fig. 2. Four instances of resource's produce-and-use features arranged chronologically relative to the September 2022 feature of "Decline meeting invitation with message": The actual testing results show that **(a)** and **(d)** turn out to be FIs, whereas **(b)** and **(c)** are non-negatively-interfering features.

TABLE I
TESTING PATTERN OF RESOURCE'S PRODUCE-AND-USE FI

testing resource-producing feature_1
  // inputs and steps to test feature_1
  … …
  … …
  resource produced ← goal_1

testing resource-using feature_2 with feature_1-produced resource
  // inputs and steps to test feature_2
  … …
  … …
  goal_2

TABLE II
TESTING PATTERN OF RESOURCE'S STATE-CHANGING FI

testing feature_1
  // inputs and steps to test feature_1
  … …
  … …
  resource's state changed successfully by feature_1

  testing feature_2 upon the state change
    // inputs and steps to test feature_2
    … …
    … …

    $\text{goal}_1 \wedge \text{goal}_2$

feature_2 stimulated with the resource outputted from feature_1, but feature_1 must also be informed by feature_2. For example, a decline message of: "I'll call you back." will *not* trigger the "Happy Anniversary chat celebration" feature, but the message: "I'll call you back. BTW, Happy Anniversary!" will. Our testing of Zoom, conducted in November 2022, showed that FIs occurred in Fig. 2-a and Fig. 2-d. The other two instances showed no FIs, i.e., the chat preview notifications were successfully made (Fig. 2-b), and the raining Tada emojis were observed (Fig. 2-c).

B. State-Changing

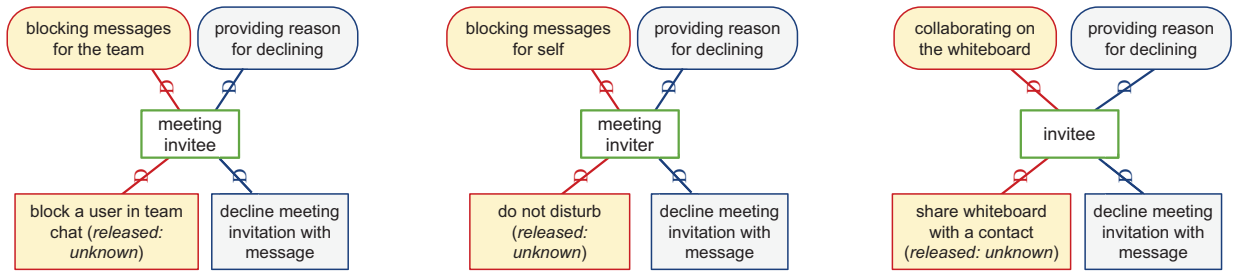
Consider a door's mechanical locking bar as a resource, and two common features operating the door: "close the door" in order to "reduce the outside noise", and "lock the door" to "prevent theft". Let us further consider two discrete states of the resource: the locking bar being "pulled out" in a locked position, and it being "backed in" in an unlocked position. Assume the initial state of the locking bar is "backed in", we can then "close the door" first followed by invoking the feature to "lock the door". This order of invocations clearly ensures that both goals of reducing the outside noise and preventing theft are met. However, if the invocation order is reversed, then the "lock the door" feature will change the resource's state to "pulled out", which in turn obstructs the "close the door" feature. Such a state change of the shared resource causes neither goal to be met, and hence an FI occurs.

We classify this type of FIs as state-changing. Different from produce-and-use, no new resource is generated. Instead, an existing resource's state is being changed by one feature (feature_1), and after that the other feature (feature_2), or both feature_1 and feature_2, could not be carried out successfully. The testing pattern of resource's state-changing FI is shown in Table II. Special attentions are paid to the order of the features' invocations, and to the test oracle of ($\text{goal}_1 \wedge \text{goal}_2$) together rather than observing them separately at different times.

For Zoom's "Decline meeting invitation with message" feature, we manually identified three instances of state-changing FIs as shown in Fig. 3. Interestingly, each of the three instances has a different resource state space. In Fig. 3-a, a meeting invitee can be "blocked" or "invited". In Fig. 3-b, a meeting inviter can stay in "do not disturb" or stay "informed". In Fig. 3-c, an invitee can be invited to "join a meeting" or "collaborate on the whiteboard". Our actual testing showed that Fig. 3-a and Fig. 3-b led to FIs, because in Fig. 3-a, even after an invitee was blocked from sending a team chat message, the decline message from the invitee still showed up, and in Fig. 3-b, even if an inviter did not want to be disturbed, the decline message was not silenced at all. No FI was detected in Fig. 3-c, as the invitee's declining status was consistent for joining a meeting and for collaborating on the whiteboard.

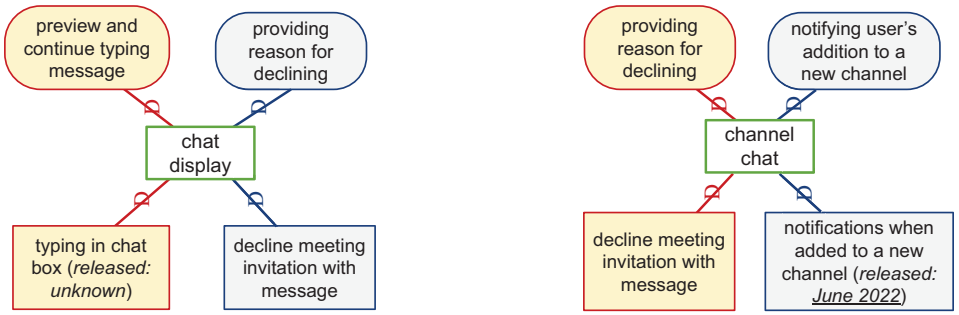
C. Mutual-Exclusion

When two features operate on the same resource concurrently, one feature can override the resource that the other feature has begun operating. For example, after the feature "recording TV program from 7pm to 8pm" has begun its operation on the video recorder (resource), some backyard movement is detected which triggers another feature "capturing burglar activities" to operate on the same resource.



(a) **(b)** **(c)**

Fig. 3. Three instances of resource's state-changing features potentially interacting with the Zoom feature of "Decline meeting invitation with message": The actual testing results show that **(a)** and **(b)** turn out to be FIs, whereas **(c)** are non-negatively-interfering features.



(a) **(b)**

Fig. 4. Two instances of resource's mutual-exclusion features interacting with the Zoom feature of "Decline meeting invitation with message": The actual testing results confirm that both are FIs.

TABLE III
TESTING PATTERN OF RESOURCE'S MUTUAL-EXCLUSION FI

```
testing feature1
    // triggering feature1's operation on resource,
    // but not completing the operation
    ... ...
    ... ...

    testing feature2 in its entirety
        // inputs and steps to test feature2
        ... ...
        ... ...
        feature2 overrides resource's state ← goal2

    // continuing the testing of feature1
    ... ...
    ... ...
    goal1
```

This later-triggered feature overrides the resource's state, or uses the resource in an exclusive fashion. As a result of this mutual-exclusion resource dependency, the previously triggered feature could not accomplish its goal.

The testing pattern of resource's mutual-exclusion FI is shown in Table III where the two features operate on the same resource in an interleaving way. Although the desired outcome is to satisfy both $goal_1$ and $goal_2$, it is likely that $goal_2$ is met due to the integrity of $feature_2$'s execution. The mutual-exclusion of the resource's concurrency control likely hurts $goal_1$ from being accomplished.

Fig. 4 presents the two mutual-exclusion instances that we identified regarding Zoom's "Decline meeting invitation with message" feature. Both instances were confirmed to be FIs. In Fig. 4-a, for instance, Alice begins typing in the chat box, "Here is today's meeting agenda: first, we will", but has *not* completed typing the message and hence has *not* sent the message yet. Alice invites Bob to join the current meeting, but Bob declines with a message. Although Alice wants to continue typing the meeting-agenda message, Bob's declining message uses the "chat display" (resource) exclusively. Alice's message gets wiped out completely. She has to re-type everything.

Fig. 4-b presents a concurrent access to another resource: channel chat. For example, Carol invites Dave to join a new channel's meeting, but Dave has *not* declined yet. Carol now asks Erin to become a member of the new channel, and Erin agrees. The notification about Erin's being a new member appears in the channel chat, completely overriding Dave's declining message even after Dave responds to Carol's invitation. From our experience, a key to the above results is to recognize the resource ("chat display" or "channel chat") more or less as a physical object that different features operate on potentially concurrently.

TABLE IV
FEATURES INTERACTING WITH ZOOM'S "ENHANCED SELF-VIEW AND NON-VIDEO PARTICIPANTS CONTROLS" FEATURE

Resource	States	Feature (release time)	Testing Result
self	{self-preview, hidden}	Blur my background (January 2022)	FI detected
	{speaker/active, hidden}	Speaker view (June 2022)	FI detected
participant	{pinned, hidden}	Multi-pin and multi-spotlight (September 2022)	FI detected
	{persistent, hidden}	Persistent gallery view (June 2022)	FI detected
	{reaction, hidden}	Raise hand for host and co-host (December 2020)	FI detected
	{main session, breakout room}	Create breakout rooms (January 2022)	FI detected
	{camera-off, camera-on, hidden, shown}	Turn camera on/off (unknown)	No FI detected

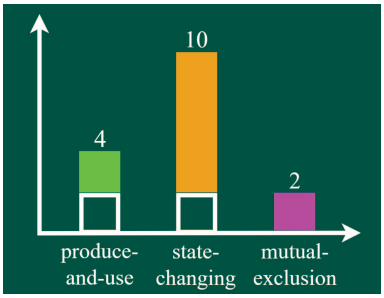


Fig. 5. Summary of the number of FIs for the two Zoom features that we analyzed: 4 produce-and-use instances were identified, among which 2 FIs (50%) were detected; 10 state-changing instances were identified, among which 8 FIs (80%) were detected; 2 mutual-exclusion instances were detected, both of which were FIs (100%).

D. Discussions

As our FI analysis and testing are performed manually, our results are limited from a completeness angle. In particular, it could be the case that more instances exist and that more types of resource-based FI exist. To address generalizability, we worked on another Zoom feature released in September 2022, "Enhanced Self-view and Non-video participants controls" [6]. The feature description is as follows:

> "The **Hide Self View** option in client settings is persistent across meetings, reducing the need to be set each meeting. Additionally, the **Hide Self View** and **Hide Non-Video Participants** options are included in the in-meeting **View** controls, for easier access."

We manually identified seven features potentially interacting with the above feature. All of the seven are of the state-changing type. Table IV lists the resources underlying the interactions, the resources' states, the interacting features, and the FI testing results. Through this analysis, we were able to confirm the testing pattern of Table II. In addition, we found that the resources were readily identifiable, as "self" and "participant" were directly mentioned in the feature's description. In fact, these resources were also highlighted in the feature's title. The states of the resources are centered around "hidden" as it was the main effect of the feature.

Analyzing the "Enhanced Self-view and Non-video participants controls" feature offered us some deeper insights into FI detection. This feature does not produce any *new* resource, and hence our current analysis was unable to identify any FIs of the produce-and-use type. Moreover, no mutual-exclusion FIs were found and our speculation is that the feature functioned *instantly* on the resource, i.e., hid self view or hid non-video participants in an atomic motion, leaving no room for another feature's concurrent execution. We anticipate observations, such as new resources and instant functions, will support the FI detection and classification more effectively and accurately; investigating these aspects is part of our ongoing work.

We summarize the number of FIs for the two Zoom features that we analyzed in Fig. 5. We emphasize that the detected FIs depend on the software's actual implementation, and thus what is essential is our ability to use goal model slices to identify the 4, 10, and 2 instances of produce-and-use, state-changing, and mutual-exclusion resource dependency respectively. With the identified instances, we can then apply the testing patterns to confirm FIs. Because software evolves continuously, re-checking all the instances in subsequent releases can be part of the regression testing activities [13]. Nevertheless, the numbers shown in Fig. 5 indicate that FIs do occur in a real-world app relied upon by many people—an encouraging result of our research at its current stage.

IV. RELATED WORK

Several taxonomies exist for FIs. Calder *et al.* [4] focused on FIs in telecommunication services and grouped the techniques into offline, online, and formal methods. Shehata *et al.* [14] defined six main interaction categories, three of which explicitly involved *resource*. In fact, Bisbal and Cheng [8] noted that *resource contention* is at the core of the FI problem, and this view was echoed by Nhlabatsi *et al.* [2] who considered that FI primarily arises from sharing of resources. Our analysis, grounded in Zoom features, leads to three novel, finer-grained types of how a resource is shared between two features and what testing steps can be undertaken to reveal the FIs.

DeVries and Cheng [15] were among the first to use a goal model to detect FIs. They not only incorporated features into the goal-decomposition hierarchy, but also added features' pre-condition and post-condition expectations to the goal model manually. These expectations were then represented in logical expressions, so that their satisfactions could be checked with a satisfiability modulo theories (SMT) solver. In the case of FIs, counterexamples were returned to help the system designer to resolve the FIs. In DeVries and Cheng's approach [15], a relatively well-constructed goal model serves as the inputs

for detecting FIs. In contrast, our approach takes the natural language (NL) descriptions of features released by software vendors as inputs, and constructs a goal model slice by modeling only the interacting features' resource dependency. We therefore advocate a lightweight, minimalist way of goal modeling in order to identify FIs.

One manifestation of the lightweigt-ness is our analysis of *pairwise* features, also known as 2-way interactions in the literature [5], [15]. Beyond the simple 2-way situations are the 3-way (or N-way) interactions which occur only when three (or N) features are composed but not when two (or $N-1$) of them are. Hence, the complexity of detecting N-way FIs is exponential, making it computationally intractable [7], [15]. An important finding of functional FIs was made by Fantechi *et al.* [16], who proved that the behavioral interactions of 3 (or N) features are always due to the interaction between 2 of the 3 (or N) considered features, thereby reducing the FI detection's complexity to pairwise analysis. For a to-be-released feature, performing pairwise FI analysis is thus linear. However, a subtlety in our approach is the recognition of relevant resource(s), along with the modeling of the resource-dependency type(s) and the execution of associated test(s). These aspects bring additional complexity which we will address as part of our long-term research plans.

V. Conclusions

Building on the preliminary results reported in this paper, we plan to develop automated support by leveraging natural language processing (NLP) techniques. A specific objective is to tag the resource(s) mentioned in a feature's description [17], and if no resource is explicitly mentioned, then we aim to infer the resource based on the feature's actions and the dependency from similar actions. Correlating features with nonfunctional requirements [18] may also be helpful.

Another support will be on retrieving candidate interacting features and classifying resource dependency types. Observations made in our current analysis, such as whether a new resource is produced and whether the operation on the resource is atomic, can help build the classifiers with explanability [19], [20]. We advocate the use of real-world software systems and applications to carry out empirical investigations, which can further facilitate replication [21], [22] and increase research's potential impacts. Our long-term goal is therefore to improve the work's relevance, scalability, and practicality.

Acknowledgment

This research is partially supported by the United States National Science Foundation (Award No. 2236953).

References

[1] Z. Peng, P. Rathod, N. Niu, T. Bhowmik, H. Liu, L. Shi, and Z. Jin, "Testing software's changing features with environment-driven abstraction identification," *Requirements Engineering*, vol. 27, no. 4, pp. 405–427, December 2022.

[2] A. Nhlabatsi, R. C. Laney, and B. Nuseibeh, "Feature interaction as a context sharing problem," in *International Conference on Feature Interactions in Software and Communication Systems (ICFI)*, Lisbon, Portugal, June 2009, pp. 133–148.

[3] N. Niu, S. Brinkkemper, X. Franch, J. Partanen, and J. Savolainen, "Requirements engineering and continuous deployment," *IEEE Software*, vol. 35, no. 2, pp. 86–90, March/April 2018.

[4] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: A critical review and considered forecast," *Computer Networks*, vol. 41, no. 1, pp. 115–141, January 2003.

[5] S. Apel, S. S. Kolesnikov, N. Siegmund, C. Kästner, and B. Garvin, "Exploring feature interactions in the wild: The new feature-interaction challenge," in *International Workshop on Feature-Oriented Software Development (FOSD)*, Indianapolis, IN, USA, October 2013, pp. 1–8.

[6] Zoom, "Release Notes for Windows," https://support.zoom.us/hc/en-us/articles/201361953-Release-notes-for-Windows, 2023, Last accessed: June 13, 2023.

[7] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The feature interaction problem in telecommunications systems," in *International Conference on Software Engineering for Telecommunication Switching Systems (SETSS)*, Bournemouth, UK, July 1989, pp. 59–62.

[8] J. Bisbal and B. H. C. Cheng, "Resource-based approach to feature interaction in adaptive software," in *Workshop on Self-Managed Systems (WOSS)*, Newport Beach, CA, USA, Oct-Nov 2004, pp. 23–27.

[9] M. Jackson, "The meaning of requirements," *Annals of Software Engineering*, vol. 3, pp. 5–21, 1997.

[10] Z. Peng, P. Rathod, N. Niu, T. Bhowmik, H. Liu, L. Shi, and Z. Jin, "Environment-driven abstraction identification for requirements-based testing," in *International Requirements Engineering Conference (RE)*, Notre Dame, IN, USA, September 2021, pp. 245–256.

[11] E. Yu, "Towards modeling and reasoning support for early-phase requirements engineering," in *International Symposium on Requirements Engineering (RE)*, Annapolis, MD, USA, January 1997, pp. 226–235.

[12] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[13] Z. Peng, X. Lin, M. Simon, and N. Niu, "Unit and regression tests of scientific software: A study on SWMM," *Journal of Computational Science*, vol. 53, pp. 101 347:1–101 347:13, July 2021.

[14] M. Shehata, A. Eberlein, and A. O. Fapojuwo, "A taxonomy for identifying requirement interactions in software systems," *Computer Networks*, vol. 51, no. 2, pp. 398–425, February 2007.

[15] B. DeVries and B. H. C. Cheng, "Automatic detection of feature interactions using symbolic analysis and evolutionary computation," in *International Conference on Software Quality, Reliability and Security (QRS)*, Lisbon, Portugal, July 2018, pp. 257–268.

[16] A. Fantechi, S. Gnesi, and L. Semini, "Optimizing feature interaction detection," in *Joint International Workshop on Formal Methods for Industrial Critical Systems and International Workshop on Automated Verification of Critical Systems (FMICS-AVoCS)*, Turin, Italy, September 2017, pp. 201–216.

[17] J. Zhang, S. Chen, J. Hua, N. Niu, and C. Liu, "Automatic terminology extraction and ranking for feature modeling," in *International Requirements Engineering Conference (RE)*, Melbourne, Australia, August 2022, pp. 51–63.

[18] N. Niu, Y. Yu, B. González-Baixauli, N. Ernst, J. Leite, and J. Mylopoulos, "Aspects across software life cycle: A goal-driven approach," *Transactions on Aspect-Oriented Software Development*, vol. VI, pp. 83–110, 2009.

[19] F. Dalpiaz and N. Niu, "Requirements engineering in the days of artificial intelligence," *IEEE Software*, vol. 37, no. 4, pp. 46–53, July/August 2020.

[20] N. Maltbie, N. Niu, M. Van Doren, and R. Johnson, "XAI tools in the public sector: A case study on predicting combined sewer overflows," in *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, Athens, Greece, August 2021, pp. 1032–1044.

[21] N. Niu, A. Koshoffer, L. Newman, C. Khatwani, C. Samarasinghe, and J. Savolainen, "Advancing repeated research in requirements engineering: a theoretical replication of viewpoint merging," in *International Requirements Engineering Conference (RE)*, Beijing, China, September 2016, pp. 186–195.

[22] C. Khatwani, X. Jin, N. Niu, A. Koshoffer, L. Newman, and J. Savolainen, "Advancing viewpoint merging in requirements engineering: A theoretical replication and explanatory study," *Requirements Engineering*, vol. 22, no. 3, pp. 317–338, September 2017.