2021 IEEE 29th International Requirements Engineering Conference (RE) | 978-1-6654-2856-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/RE51729.2021.00029

Environment-Driven Abstraction Identification for Requirements-Based Testing

Zedong Peng
University of Cincinnati
Cincinnati, OH, USA
pengzd@mail.uc.edu

Prachi Rathod
University of Cincinnati
Cincinnati, OH, USA
rathodpt@mail.uc.edu

Nan Niu
University of Cincinnati
Cincinnati, OH, USA
nan.niu@uc.edu

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

Hui Liu
Beijing Institute of Technology
Beijing, China
liuhui08@bit.edu.cn

Lin Shi
Institute of Software Chinese Academy of Sciences,
University of Chinese Academy of Sciences, China
shilin@iscas.ac.cn

Zhi Jin
Peking University
Beijing, China
zhijin@pku.edu.cn

***Abstract*—Abstractions are significant domain terms that have assisted in requirements elicitation and modeling. To extend the assistance towards requirements validation, we present in this paper an automated approach to identifying the abstractions for supporting requirements-based testing. We select relevant Wikipedia pages to serve as a domain corpus that is independent from any specific software system. We further define five novel patterns based on part-of-speech tagging and dependency parsing, and frame our candidate abstractions in the form of <key, value> pairs for better testability. We evaluate our approach with six software systems in two application domains: Electronic health records and Web conferencing. The results show that our abstractions are more accurate than those generated by two of the state-of-the-art techniques. Initial findings also indicate our abstractions' capabilities of revealing bugs and matching the environmental assumptions created manually.**

***Index Terms*—abstractions, natural language, environmental assumptions, requirements-based testing**

I. Introduction

In requirements engineering (RE), an *abstraction* refers to a term that has a particular significance in a given domain [1]. For example, "radar" is recognized as an abstraction in the air traffic control problem domain [2], and so is "antenna" in the radio frequency identification (RFID) application domain [3]. In order to reduce the requirements engineer's effort, researchers have developed methods to automatically identify the abstractions from the natural language (NL) documents. While the seminal work of AbstFinder searches for patterns of byte sequences [4], other researchers have located the abstraction candidates by exploiting natural language processing (NLP) techniques (e.g., corpus-based frequency profiling and part-of-speech tagging) [2], [3], [5]–[7].

Current support is mainly for *early phase RE* where the focus is on understanding the problem domain *before* formulation of the initial requirements [8]. For instance, Sawyer *et al.* [2] showed in an air traffic control case study that the abstractions extracted from a set of ethnographic fieldnotes by NLP could match the elements of a class diagram at a 75% recall and 12% precision level. Clearly, the relevance of such NLP results must be vetted by the requirements engineer.

Indeed, Ryan [9] argued that NLP should play only a partial role in requirements validation, i.e., demonstrating convincingly a software system's conformance to stakeholder needs, because validating requirements must remain an informal, social process. Ryan [9] further pointed out that an intrinsic difficulty lies in the identification of *assumptions* that reflect the shared ("common sense") knowledge of people familiar with the social and technical contexts within which the software system operates.

Significant to RE are the *environmental assumptions* [10], i.e., the conditions over the phenomena of the physical world that one accepts as true irrespective of the software to be built [11]. In this paper, we use "assertion" [11] and "assumption" [12] interchangeably to refer to a *statement* indicating a property over the phenomena in the software's operational context that is accepted as true by the developers [13]. Many software problems originate in missing or flawed environmental assumptions.

A recent empirical study by Bhowmik *et al.* [14] with 114 developers showed the positive impact of environmental assumptions on requirements-based testing. One concrete result highlighted the assumption: "a doctor's appointment shall be scheduled only for a future timeslot". This statement generally holds independent of any specific software system. As a result, it helped to uncover a defect in a software application where a patient was able to make an appointment for a past date and time [14]. Despite the positive impact, Bhowmik *et al.* [14] reported that manually formulating complete and correct environmental assumptions from scratch is challenging.

Using NLP to automatically produce assumption statements, according to Ryan [9], is infeasible; however, narrow domain understanding in the form of abstractions has been shown to be realistic [1]–[4]. Our objective in this paper is to automatically identify the abstractions that are both indicative of important domain phenomena and amenable to requirements-based testing. To that end, we derive a corpus by selecting pages from Wikipedia, an exceptional repository codifying our shared knowledge about specific and connected topics.

We then define a novel set of NLP patterns to extract and rank candidate abstractions. We show the effectiveness of our approach by comparing its results with abstractions identified from the state-of-the-art methods [1], [4]. We also demonstrate our approach's usefulness by relating the resulting abstractions to the environmental assumptions created manually [12], [14].

The main contribution of our work is the abstraction identification that goes beyond early phase RE and offers new support for requirements validation. Our evaluation with six software applications in two different domains shows the effectiveness of our approach. In what follows, we present background information in Section II. We then clarify the influence of environmental assumptions on requirements-based testing in Section III. Section IV details our NLP tool chain, Section V describes the empirical evaluations, and finally, Section VI concludes the paper.

II. Background and Related Work

A. Abstraction Identification in Requirements Engineering

Abstractions are important domain concepts which the human analyst needs to identify in order to understand the problem domain as well as the constraints on the range of possible solutions [2]. To support the elicitation of the initial requirements, Goldin and Berry [4] developed the AbstFinder tool based on the idea that important abstractions would recur frequently as repeated words within the target NL document. AbstFinder thus searches for co-occurring byte sequences within pairs of sentences using a series of circular shifts, and returns a ranked list of frequently occurring words and phrases.

To be successful in supporting early phase RE, the automatic identification of abstractions must achieve a level of completeness at least as good as that achieved by a human analyst. High recall values are often obtained at the cost of low levels of precision [15]–[17]. As noted in [4], when a complex problem is tackled, a precision of 25% or higher represents good abstraction identification performance.

Sawyer *et al.*'s air traffic control case study confirmed the practically achievable precision level, where a NLP toolset was used to extract abstractions from an aggregated set of ethnographic fieldnotes comprising about 44,000 words [2]. The technique achieved a 21% precision, showing the practical performance when processing a sizable volume of text.

Gacitua *et al.* [1], [3] used corpus-based frequency profiling to identify single-word abstractions. Given a domain document D and a normative corpus C, frequency profiling computes a term t's log-likelihood value[1] LL_t according to t's observed values in D and its expected values in C. The greater the LL_t value is, the more significant t is in D than in C, and hence the more likely t is an abstraction. Because over 85% domain-specific terms are multiword units [18], Gacitua *et al.* also recognized multi-word abstractions via syntactic patterns based on PoS tagging, namely "adjectives and nouns", and "adverbs and verbs". In an experiment with the full text of a book containing 156,028 words, Gacitua *et al.*'s method achieved a 32% recall and a 32% precision, outperforming AbstFinder's 7% recall and 7% precision [1].

[1]Mathematical definition is provided later in equation (6) of Section V-B.

In summary, abstraction identification can be thought of as where the expertise of domain expert and requirements engineer meet [3]. Since domain expertise is often available to the requirements engineer as NL documents (e.g., marketing reports), abstractions identified from the relevant documents help encapsulate the rich contextual information needed for the framing of the requirements. Framing the requirements, as Jackson [11] pointed out, must consider explicitly the phenomena of the environment in which the software operates.

B. Environmental Assumptions

An assumption is defined as: "*a thing that is accepted as true or as certain to happen, without proof*"[2]. In software development, an *environmental assumption* is a statement about the software system's operational context that is accepted as true by the developers [13]. For example, the statement: "a train is moving if and only if its speed is non-null" [12] is an assumption made about the physical world. Many software problems originate in missing, inadequate, inaccurate, or changing environmental assumptions [12]. Notably, the assumption made about the maximum horizontal velocity did not hold for Ariane 5, contributing to the rocket launch failure [19].

Diagnosing whether an existing assumption is incorrectly removed or retained in a software product line is addressed by Rahimi and her colleagues [20]. This diagnostic approach considers only the safety-related assumptions that are linked to the manually conducted Failure Mode, Effects and Criticality Analysis (FMECA) [21]. Rules are then defined to flag potential assumption errors for a new or changed product.

Although documented assumptions may be flawed, one of the most critical problems is that assumptions are usually kept undocumented in software projects [22], leading to architectural mismatches [23], budget and schedule overruns [24], security vulnerabilities [25], and a multitude of system issues, defects, and failures. Similar to requirements and source code, assumptions are a type of software artifacts being produced, modified, and used by a process [26]. Next we present our proposal for integrating environmental assumptions in the requirements-based testing process.

III. Environmental Assumptions and Requirements-Based Testing

Skoković and Skoković [27] introduced requirements-based testing (RBT) to address two major issues in software quality assurance: (1) validating the requirements are unambiguous, consistent, and complete, and (2) designing a necessary and sufficient set of test cases from a black-box perspective to cover the validated requirements. Fig. 1 shows the three RBT activities, which we use iTrust's[3] "Schedule Appointments"

[2]http://www.oxforddictionaries.com/definition/english/assumption

[3]iTrust is a Java application that provides patients with a means to keep up with their medical records and to communicate with their doctors [28].

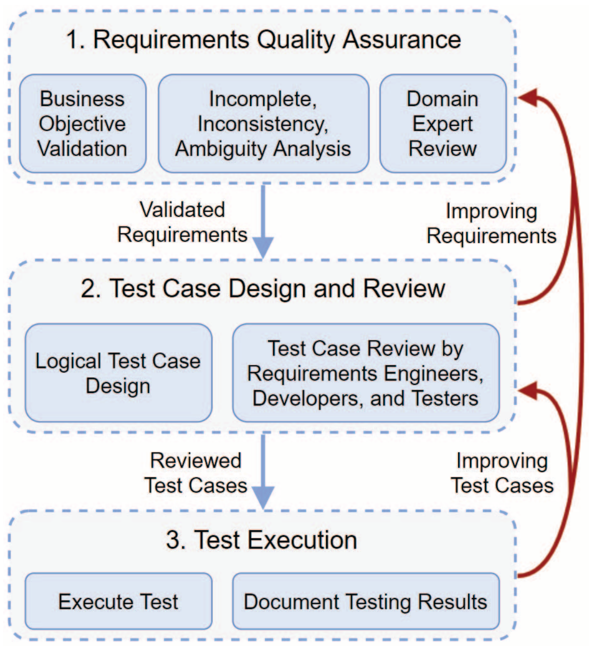

1. Requirements Quality Assurance
Business Objective Validation
Incomplete, Inconsistency, Ambiguity Analysis
Domain Expert Review
Validated Requirements
Improving Requirements
2. Test Case Design and Review
Logical Test Case Design
Test Case Review by Requirements Engineers, Developers, and Testers
Reviewed Test Cases
Improving Test Cases
3. Test Execution
Execute Test
Document Testing Results

Fig. 1. The RBT process flow (*adapted from [27]*).

requirement [28] to illustrate. Fig. 2 displays a snippet of this use case.

RBT's first activity of requirements quality assurance stands out from other traditional testing techniques. Not only must the requirements be validated against the business objectives, but an initial review shall be conducted to try to find errors in requirements, such as ambiguity, incompleteness, and inconsistency [29].

Given the validated requirements, the next RBT activity is to design such black-box, logical test cases as to achieve high test coverage of the requirements [27]. Designing and reviewing logical test cases can help discover requirements problems, e.g., E1, as currently stated in Fig. 2, is triggered by a new appointment type's name over 30 characters, or by a duration unit not entered in minutes. However, S1 lacks information about whether the name and the duration are mandatory at a new appointment type's creation time. Thus, the requirements can be clarified for better logical test coverage.

The third RBT activity shown in Fig. 1 concerns executing tests by adding data to the logical test cases. Once all of the tests execute successfully against the code, Skoković and Skoković [27] argue that 100% of the functionality has been verified and the code is ready to be delivered into production.

Despite RBT's attentions paid to requirements, we believe the process of Fig. 1 can be enhanced by two shifts, both involving environmental assumptions. First and foremost, the meaning of requirements is defined by Jackson as [11]:

$$\mathcal{E}, \mathcal{S} \vdash \mathcal{R} \tag{1}$$

where $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ represent environmental assumptions, specifications, and requirements respectively. Fig. 3 depicts the

Use Case: Schedule Appointments

Sub-flows:

[S1] The system shall enable the administrator to add a new entry for an appointment type, including its type name with up to 30 alpha characters and duration in the unit of minutes [E1].

[S2] The LHCP (licensed health care professional) schedules an appointment with a patient, and enters comment (optional) up to 1000 characters [E2].

[S3] The patient selects an LHCP from his or her provider list. The patient selects the type of appointment, enters the appointment date and start time. If the requested appointment time does not conflict with any existing appointment for the LHCP, the request is saved. If the requested appointment time does conflict with an existing appointment, the patient is presented with a list of the three next non -overlapping available appointment times within 7 days of the requested date. The patient selects one of these appointments and the request is saved.

Alternative Flows:

[E1] The user inputs invalid information and is prompted to try again.

[E2] The comment is empty and the text "No Comment" (without link) is displayed instead of the "Read Comment" link.

Fig. 2. Snippet of iTrust's "Schedule Appointments" use case (*adapted from [14]*).

conceptual distinction and overlap between the environment and the machine (software-to-be). A customer requirement $\mathcal{R}$ expresses a condition over the phenomena of the environment that we wish to make true by installing the machine, whereas an environmental assumption $\mathcal{E}$ expresses a condition over the phenomena of the environment that we accept to be true irrespective of the properties and behavior of the machine [11]. The $\vdash$ among $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{R}$ is an entailment. Compared with validating $\mathcal{R}$ against business objectives shown in the top activity of Fig. 1, equation (1) shifts requirements validation toward the entailment relationship, $\vdash$, to which $\mathcal{E}$ is integral.

In Fig. 3, $\mathcal{P}$ and $\mathcal{C}$ are private to the machine domain: $\mathcal{P}$ denotes the program implementing the specification, and $\mathcal{C}$ denotes the computing platform on which $\mathcal{P}$ runs. The correctness of a software implementation is given as [30]:

$$\mathcal{P}, \mathcal{C} \vdash \mathcal{S} \tag{2}$$

and we further denote the software under test (SUT) by $(\mathcal{P}, \mathcal{C})$ to indicate that software testing, including RBT, is to stimulate $\mathcal{P}$ on $\mathcal{C}$ as a whole with concrete input data. From equations (1) and (2), we have:

$$\mathcal{E}, \mathrm{SUT} \vdash \mathcal{R} \tag{3}$$

deducing requirements validation in the absence of $\mathcal{S}$.

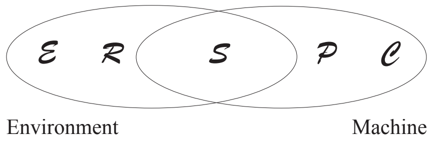


Fig. 3. The environment is the part of the world with which the machine (i.e., the software-intensive system) will interact (*adapted from [11] and [30]*).

The second shift that we propose to the RBT process shown in Fig. 1 is to highlight the practical value of testing. Dijkstra famously said, "*Testing shows the presence, not the absence of bugs*"[4]. We argue that searching for those environmental assumptions and test inputs such that:

$$\mathcal{E}, \mathrm{SUT} \nvDash \mathcal{R} \tag{4}$$

would be more valuable to RBT than trying to achieve a 100% test coverage against the requirements. Let us revisit iTrust's "Schedule Appointment" requirement in Fig. 2. Subflow [S3] describes two branches after the patient enters the requested appointment time: no time conflict with the LHCP or otherwise. Two logical test cases can then be designed to have both branches covered. However, an unstated assumption about [S3] is that: "If a doctor's appointment cannot be made at the patient's preferred time, the patient will accept an alternative appointment time within 7 days of the preferred time."

Making this assumption explicit allows us to construct a concrete $\mathcal{E}$ (e.g., "a patient wants to know the options, and possibly schedules the appointment, beyond 7 days of the unfulfilled, preferred time") such that $\mathcal{E}$, $\mathrm{SUT} \nvDash \mathcal{R}$, effectively showing the presence of a bug[5] and provoking requirements changes. From this example, we derive the desiderata of the kinds of $\mathcal{E}$ that best support our revised RBT process:

- $\mathcal{E}$ shall be **machine-independent**. Regardless of iTrust or any other machine being the SUT, $\mathcal{E}$ is in the indicative mood [11], expressing what is assumed to be true in the environment.
- $\mathcal{E}$ shall be **requirements-related**. Although $\mathcal{E}$ is independent of the machine, it should not be indifferent to $\mathcal{R}$. Even for the same SUT, different requirements, in principle, need different assumptions to support the RBT.
- $\mathcal{E}$ shall be **directly-testable**. The $\mathcal{E}$ that gives rise to executable tests is preferred. In addition to being closely related to $\mathcal{R}$, $\mathcal{E}$ shall induce as concrete test inputs as possible to trigger the SUT.
- $\mathcal{E}$ shall be **bug-revealing**. The practical value of showing the presence of bugs implies that the RBT helps to uncover flawed assumptions, faulty implementations, or invalidated requirements.

Automatically generating $\mathcal{E}$ with all the desiderata from NL documents is unrealistic [9], but the less ambitious goal of assisting the discovery of problem domain properties is feasible, as evidenced by abstraction identification in RE (cf. Section II-A). The next section presents our automated support for identifying abstractions driven by the desiderata of $\mathcal{E}$.

[4]https://en.wikiquote.org/wiki/Edsger_W._Dijkstra

[5]By "bug", we mean equation (4) is evaluated to be true (i.e., the entailment relationship fails to hold) under a specific set of $\mathcal{E}$, SUT, and $\mathcal{R}$.

IV. Abstraction Identification for Environmental Assumptions

Fig. 4 shows an overview of our NLP tool chain for extracting and ranking abstractions. The candidate abstractions are then used to assist the human analyst in performing the RBT. Our approach consists of three major steps: corpus selection from Wikipedia, abstraction identification via NLP patterns, and abstraction ranking according to the textual similarity between the extracted abstractions and the NL requirements of a software-intensive system. This section uses iTrust [28], a software system helping to manage electronic health records, to illustrate our approach.

A. Preprocessing

The data source that we use to identify the abstractions is Wikipedia. The main rationale is to achieve the **machine-independent** property of the resulting abstractions. Wikipedia is a vast online source of human knowledge that describes the concepts across a wide range of domains. These descriptions are independent from any specific software solutions, and are indicative stating what is believed to be true rather than what is wished to be true by introducing a software solution. Additionally, Wikipedia is a text corpus that collectively contains the current conventional wisdom of the subject matters [31]. This ensures the descriptions are less biased than individuals' subjective opinions (e.g., tweets about a software product).

To select a corpus for our purpose of abstraction identification, a seed page is required to anchor the domain interests. Our approach relies on the human analyst to provide such a seed page. For iTrust, for example, the Wikipedia page on "Electronic health records"[6] is manually chosen as the seed page, from which a corpus containing related pages is derived. We build on Ezzini *et al.*'s recent work where they used a domain-specific corpus for detecting requirements ambiguities [32]. While using a corpus that is too small would be ineffective in recognizing significant terms and their relationships, building and using a corpus that is too large would be time-consuming, and more importantly, would defeat the goal of being domain-specific [32]. In Ezzini *et al.*'s work, 50–250 keywords were tested, and in our work here, we empirically set the corpus's size to be 250 Wikipedia pages.

To grow from the single seed page to the 250-page corpus, we distinguish two kinds of Wikipedia pages: *content page* introducing a topic and *category page* listing a set of content pages belong to a topic as well as the sub-categories of the topic. Fig. 5 illustrates the distinction: the sub-figure to the left is a content page whereas the one to the right is a category page, showing that "Electronic health records" contain two sub-categories and 45 pages. We further note the hyperlinks within each content page, e.g., "health care" and "information

[6]https://en.wikipedia.org/wiki/Electronic_health_record

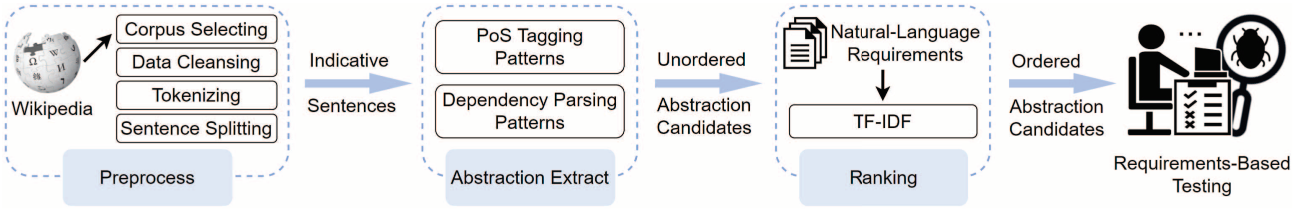


Fig. 4. Components of the NLP-aided abstraction identification approach.

systems" of Fig. 5a. These links not only provide an efficient navigation mechanism over the Wikipedia contents, but also represent some semantic relationships between pages or categories [33]. Our corpus is constructed with the following procedure: While the total number of pages is less than 250,

1) Add the manually identified seed page, resulting in the "Electronic health records" corpus size to be 1 page;
2) Add all the content pages belong to the seed page's topic, making the corpus's size grow to (1+44) = 45 pages;
3) Add all the content pages belonging to the sub-categories of the seed page's topic, leading to a (45+30+30) = 105-page corpus; and
4) From the already added content pages, add the Wikipedia content pages of the hyperlinks *before* each page's structured table of contents (cf. Fig. 5a). The reason that we include into our corpus only the hyperlinked pages before the table of contents is because these topics provide substantial background information for the main topic of interest. We operate 4) based on when a page is added by following the above 1), 2) and 3) ordering, till a total of 250 Wikipedia pages is reached.

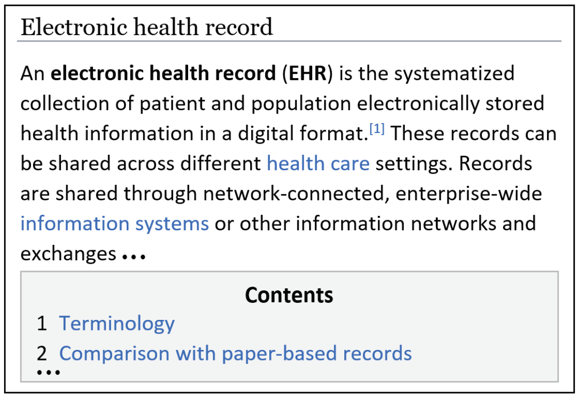
We implemented our page selection logic by using the Beautiful Soup Python library [34]. Our Python-based corpus builder also ensured that no duplicate Wikipedia page was selected. Once the corpus's 250 pages were chosen, Fig. 4 shows that data cleansing, tokenizing, and sentence splitting would take place. For each page, our data cleansing removed the figures and the formatting information (e.g., table of contents). Following prior work [35], we then applied spaCy's tokenizer [36] to break each page into tokens: words, numbers, punctuation marks, or symbols. Finally, we used spaCy's sentencizer [36] to split the text into sentences based on conventional delimiters (e.g., period).

B. Extracting Abstractions

We process each sentence from the selected Wikipedia pages in order to find **directly-testable** abstraction candidates. We operationalize "testability" by formatting an abstraction as a <key, value> pair. This hash structure is intended to separate a domain concept (key) from its manifestation (value). We expect the "key" to help locate "what to test", and the "value" to guide "how to test it" by feeding in concrete data.

Building on the NLP-based abstraction identification approaches [2], [3], we define five patterns by exploiting the syntactic and grammatical roles that words play in a sentence. Fig. 6 lists these patterns. Note that Gacitua *et al.* [1], [3] used two PoS patterns—"adjectives and nouns" and "adverbs and verbs"—to identify abstractions; however, the two PoS patterns of our approach, PoS_P1 and PoS_P2 shown in Fig. 6, consider <key, value> explicitly.

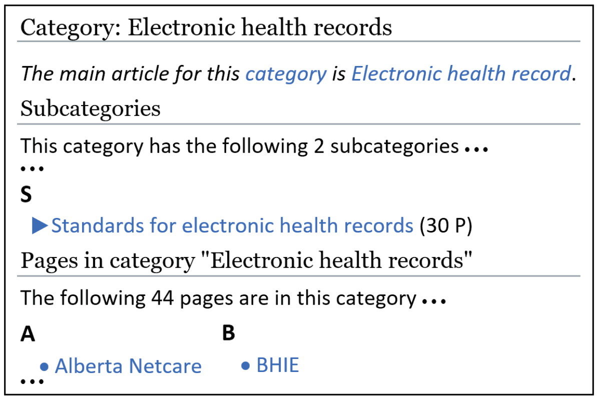
Dependency parsing [37] is the task of identifying the grammatical structure of a sentence by determining the linguistic dependencies between the words based on a predefined set of *dependency types*. For example, in the sentence: "The system shall refresh the display", "display" is the direct


Electronic health record

An **electronic health record** (**EHR**) is the systematized collection of patient and population electronically stored health information in a digital format.[1] These records can be shared across different health care settings. Records are shared through network-connected, enterprise-wide information systems or other information networks and exchanges ...

Contents
1 Terminology
2 Comparison with paper-based records
...

(a) Content page


Category: Electronic health records

The main article for this category is Electronic health record.

Subcategories

This category has the following 2 subcategories ...
...
S
▶Standards for electronic health records (30 P)

Pages in category "Electronic health records"

The following 44 pages are in this category ...

A	B
• Alberta Netcare	• BHIE
...

(b) Category page

Fig. 5. Sample content page (used in our domain corpus) and sample category page (used to find more content pages to be included in the corpus).

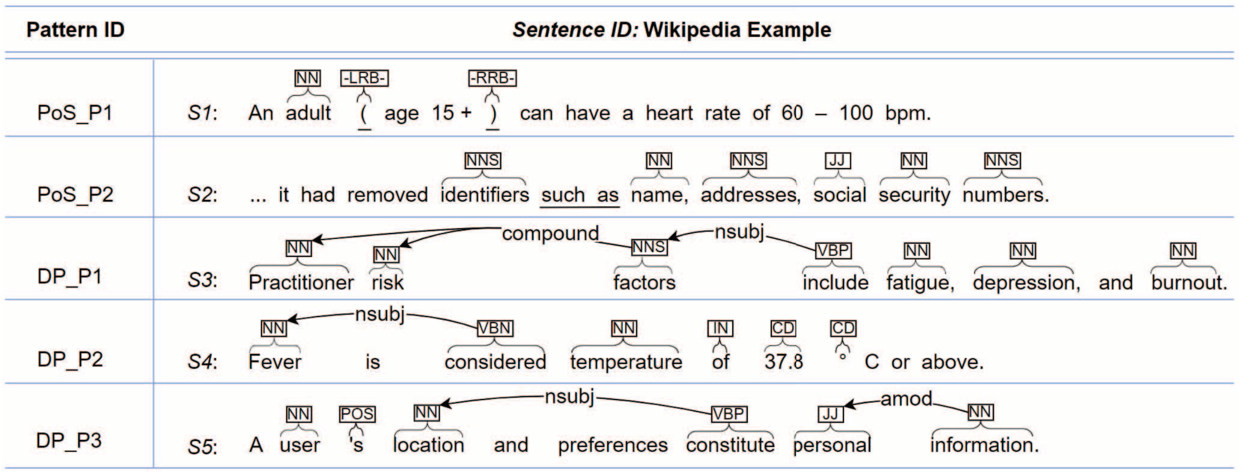

Pattern ID
Sentence ID: Wikipedia Example
PoS_P1
S1: An adult (age 15 +) can have a heart rate of 60 – 100 bpm.
NN -LRB- -RRB-
PoS_P2
S2: ... it had removed identifiers such as name, addresses, social security numbers.
NNS NN NNS JJ NN NNS
DP_P1
S3: Practitioner risk factors include fatigue, depression, and burnout.
compound nsubj
NN NN NNS VBP NN NN NN
DP_P2
S4: Fever is considered temperature of 37.8 ° C or above.
nsubj
NN VBN NN IN CD CD
DP_P3
S5: A user 's location and preferences constitute personal information.
nsubj amod
NN POS NN VBP JJ NN

Fig. 6. Illustration of our five patterns for identifying <key, value> abstractions in a sentence. PoS (part-of-speech) tags shown here include: "NN" (noun, singular), "-LRB-" (left round bracket), "-RRB-" (right round bracket), "NNS" (noun, plural), "JJ" (adjective), "VBP" (verb, non-3rd person singular present), "VBN" (verb, past participle), "IN" (preposition), "CD" (cardinal number), and "POS" (possessive ending). Dependency parsing types shown here include: "compound" (noun compound modifier), "nsubj" (nominal subject), and "amod" (adjectival modifier).

object (*dobj*) of the main verb "refresh" whereas "shall" is the auxiliary verb (*aux*) adding modality to the main verb. Dalpiaz *et al.* [38] recently exploited dependency parsing to classify requirements. Although our objective is to identify abstractions, we believe dependency parsing, just like PoS tagging, can constitute an effective NLP toolset that offers deep domain understandings [2]. We detail each of the five patterns as follows.

- **PoS_P1** extracts the NN or NNS that precedes the parentheses as key, and the content inside the parentheses as value. Cohen *et al.* [39] showed that the parenthesized material in biomedical text often contains data value or list element useful for information extraction, which we also observe in Wikipedia pages. This pattern therefore extracts <"adult", "age 15 +"> from *S1* of Fig. 6.
- **PoS_P2** leverages a common lexico-syntactic way of providing examples [40] so as to identify the NN or NNS to be a key, and the hyponym(s) [41] following "such as" to be its value(s). The pattern thus outputs <"identifiers", "name" "addresses" "social security numbers"> as an abstraction candidate for Fig. 6's *S2*.
- **DP_P1** recognizes the nominal subject ("nsub") or the subject's compound ("compound") as a key, and the parallel NN's after the main verb "include" (or "includes") as values. Different from the "such as" pattern used in **PoS_P2**, "include" signals a part-whole relationship [42], listing several concrete facets of the concept. In Fig. 6's *S3*, <"practitioner risk factors", "fatigue" "depression" "burnout"> is identified as an abstraction candidate.
- **DP_P2** treats the nominal subject as a key in the same manner as **DP_P1**; however, its value is extracted based on the sequence of "NN, IN, and CD" appearing after the main verb. **DP_P2** is informed by Dalpiaz *et al.*'s finding that "IN and CD" often distinguished requirements types [38]. For *S4* of Fig. 6, **DP_P2** uncovers <"fever", "temperature of 37.8°"> as a candidate pair.
- **DP_P3** also considers the nominal subject to be a key, with the value identified via the NN that follows the main verb and that is modified by an adjectival ("amod"), nominal ("nmod"), or numeric ("nummod") modifier. Finin [43] noted that modifier takes a head concept and a potential modifying concept and produces a set of possible interpretations, which we adopt to derive **DP_P3**. According to this pattern, <"user's location", "personal information"> is recognized from *S5* of Fig. 6.

The above set is by no means an exhaustive list of grammatical features that must be associated with environmental-assumption statements, but a means of automatically extracting directly-testable abstractions. The NLP patterns are informed by the relevant literature [38]–[43] and further realized by our PoS tagging and dependency parsing implementations built on top of the open-source spaCy library [36] written in Python.

C. Ordering Abstractions

So far, our approach processes information related to a *domain* (e.g., Electronic health records). To ensure that our abstraction identification is **requirements-related**, we use the NL requirements of each specific *product* [44] to rank the abstraction candidates. In this way, even though some software-intensive systems are in the same domain, their *ranked* abstractions will be different due to the differences of the requirements at the product level.

Given a product's requirements R=$\{r_1, r_2, \ldots, r_m\}$ (e.g., iTrust's 54 use cases), we first compute the cosine similarity measure between a candidate abstraction ($abst$) and R with TF-IDF weighting [45]–[47]:

$$\text{cosine}(abst, R) = \sum_{i=1}^{m} \text{cosine}(abst, r_i). \quad (5)$$

TABLE I
DOMAIN AND SUBJECT SYSTEM CHARACTERISTICS

		Electronic health records	Web conferencing
# of sentences		33,988	43,744
category depths		3	3
# (%) of sentences to which a pattern applies	PoS_P1	5,437 (16.0%)	4,553 (10.4%)
	PoS_P2	1,962 (5.8%)	1,182 (2.7%)
	DP_P1	2,133 (6.3%)	785 (1.8%)
	DP_P2	3,236 (9.5%)	2,945 (6.7%)
	DP_P3	17,393 (51.2%)	15,973 (36.5%)
# of product-level requirements		54 (iTrust)	49 (Teams)
		27 (OpenEMR)	26 (Webex)
		39 (OpenMRS)	31 (Zoom)

Note that R can be a set of selected requirements, or even a set with one requirement of interest. We then rank all abstraction candidates by their cosine similarity scores in a descending order.

V. EVALUATION

This section presents a comprehensive evaluation of our approach, comparing it with two state-of-the-art abstraction identification techniques: AbstFinder [4] and relevance-based abstraction identification (RAI) [1]. Furthermore, we examine the resulting abstractions' matching with manually created environmental assumptions, as well as their capabilities of revealing software defects. We answer three research questions (RQs) in this section by assessing the effectiveness (RQ1), partial-ness (RQ2), and bug-revealing-ness (RQ3) of the produced abstractions. All our experimental materials are publicly available at https://doi.org/10.5281/zenodo.4618108 .

A. Domains and Subject Systems

We chose to study two domains, Electronic health records and Web conferencing, due to our familiarity with them and the availability of software applications in them. We ran our Python-based preprocessing steps in February 2021. Table I shows that, when we manually selected https://en.wikipedia.org/wiki/Electronic_health_record and https://en.wikipedia.org/wiki/Web_conferencing to be the seed page respectively, a total of 33,988 and 43,744 sentences were collected. In both domains, the selected 250 Wikipedia pages were within 3 category depths of each other, implying these pages' topical closeness [48]. The five NLP patterns' applicability ranged from 1.8% to 51.2% of the sentences. In both domains, **DP_P3** had a wide influence, showing that many Wikipedia sentences have described the nominal subject ("nsub") with adjectival, nominal, or numeric modifiers.

We selected three software applications for each chosen domain. Table I lists the number of NL requirements for these applications. In addition to iTrust, we investigated two open-source medical record systems: OpenEMR [49] and OpenMRS [50]. OpenEMR is one of the most popular electronic medical records in use today with over 7,000 downloads per month, and its project repository lists 27 features, such as patient scheduling, prescriptions, and medical billing [51].

Zoom_r1:	Presenters can share their whole desktop or individual applications.
Zoom_r2:	Primary camera view will automatically toggle to the active speaker.
Zoom_r3:	Browser, client, and plugin scheduling options, including delegation for co-hosts and schedulers.
Zoom_r4:	Record meetings locally and upload to Box, OneDrive Video, or Youtube.
Zoom_r5:	Zoom sessions can be expanded to allow larger groups, up to 500 interactive participants in Large Rooms or 10,000 viewers via Zoom Webinars.
Zoom_r6:	Hide all participants whose cameras are disabled.
Zoom_r7:	Automatically transcribe the audio of a meeting or a webinar; the host can then edit the transcript.

Fig. 7. Sample NL requirements of Zoom.

OpenMRS allows for customizable electronic medical record systems to support the delivery of health care in developing countries; 39 requirements are introduced in OpenMRS's user guide [52], including viewing and creating patient records, patient dashboard, etc.

Web conferencing has become one of the most prevalent and useful tools due to the COVID-19 pandemic. We studied three popular products: Zoom, Cisco Webex, and Microsoft Teams. While our experimental materials list all the product-level requirements and the online resources from which we collected the requirements, Fig. 7 shows a few Zoom features.

B. Effectiveness of Abstraction Identification

As in [1], [4], we measure abstraction identification's effectiveness by precision and recall. In addition to our <key, value> pair (KVP) abstractions, we also experimented with AbstFinder [4] and RAI [1]. AbstFinder eschews standard text processing techniques to treat the NL documents as a stream of bytes, ignoring the lexical elements. For instance, given a stream of "user needs require an access to", AbstFinder uses a sliding window technique that causes the circular shifts of "by username and password access", as illustrated below:

```
|user| n e e d s   r e q u i r e   a n |access|  t o

|by u|s e r n a m e  a n d  p a s s w o|r d  a c c|e s s
|y us|e r n a m e  a n d  p a s s w o r|d  a c c e|s s  b
| use|r n a m e  a n d  p a s s w o r d|  a c c e s|s  b y
|user|n a m e  a n d  p a s s w o r d  |access|  b y    → user access
  ...          ...          ...
```

As a result, an abstraction candidate, "user access", would be recognized. We fed the Wikipedia pages and the product-level requirements into AbstFinder, and ranked the candidate abstractions by the frequency of occurrence.

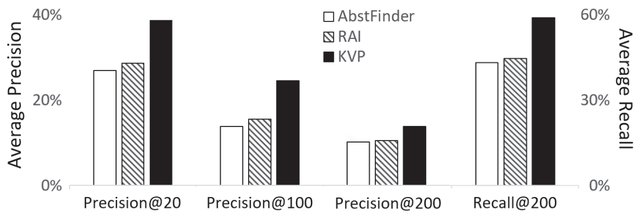
RAI uses corpus-based frequency profiling to compute the log-likelihood (LL) value for a word w [1]:

$$LL_w = 2 \cdot \left(w_d \cdot \ln \frac{w_d}{E_d} + w_r \cdot \ln \frac{w_r}{E_r} \right) \quad (6)$$

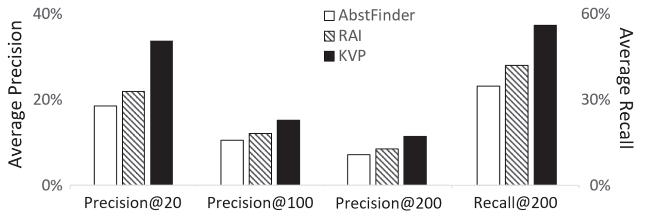
where w_d is the number of times w appears in the domain documents of the 250 Wikipedia pages, and w_r is the number of times w appears in the product-level requirements. While w_d and w_r are observed values of w, E_d and E_r in equation (6) are expected values: $E_d = \frac{n_d \cdot (w_d + w_r)}{n_d + n_r}$ and $E_r = \frac{n_r \cdot (w_d + w_r)}{n_d + n_r}$, where n_d is the total number of words in the domain documents, and n_r is the total number of words in the product-level requirements. We ranked the single-word abstraction candidates in the ascending order of LL_w to obtain testable constructs: the smaller is LL_w, the significance of w in domain documents is closer to that of w in product-level requirements, making w's ranking closer to the top. The multiword abstraction candidates were recognized by the "adjectives and nouns" and "adverbs and verbs" PoS patterns, and then ranked by an aggregated LL for the multiword unit [1].

Table II presents a comparison of AbstFinder, RAI, and our approach. Due to space constraints, we show only the top-five abstraction candidates of iTrust and OpenEMR from the healthcare domain. To quantitatively and practically evaluate different abstraction identification techniques, we measured precision of the topmost 20, 100, and 200 results. Sawyer *et al.* [2] hypothesized that human analysts would be reluctant to explore a long list of abstraction candidates and used the 20 highest ranked candidates to simulate the hypothesis. Gacitua *et al.* [1] evaluated the topmost 200 candidates as a practical upper bound. Therefore, we also computed recall@200 without exceeding this limit.

The answer set for each subject system was constructed by two researchers: first individually and then jointly to resolve discrepancies. Cohen's kappa before the joint meeting was 0.57, signifying a moderate level of inter-rater agreement. The main challenge was to handle abstractions with varying lengths yet overlapping partially, e.g., in iTrust, AbstFinder's 11th output was "imaging", RAI's 153rd output was "medical imaging", and our 31st pair was <"medical imaging", "time">. The research team decided to unify these candidates with shortest-common-supersequence [53], i.e., creating *one* element of "medical imaging time" in the answer set and then using *'contained in'* to establish a match. In this way, AbstFinder's 11th, RAI's 153rd, KVP's 31st outputs were considered as matching the answer set's element of "medical imaging time" because all three candidates were 'contained in' that element. Note that the 'contained in' match was



(a) Electronic health records



(b) Web conferencing

Fig. 8. Accuracy of candidate abstractions.

not ordered, so "time medical" would be a match, too. The two researchers collaboratively defined the answer set for each subject system based on their individual judgments and the shortest-common-supersequence unifying step. A third researcher reviewed and agreed on the answer sets, which we share as a part of our experimental materials.

Fig. 8 plots the average precision and recall among the three systems in each domain. When considering precision, we note that KVP outperforms RAI and AbstFinder. Such effects are observed more prominently in the healthcare domain than web conferencing. One reason may be the relatively homogeneous domain concepts in web conferencing; in contrast, healthcare covers wider domain phenomena (e.g., symptoms, health conditions, and treatments). Given that a precision over 25% represents practically achievable good abstraction identification performance [4], all three methods perform well at Precision@20 in the healthcare domain; however, KVP also performs well at Precision@20 in web conferencing and even at Precision@100 in healthcare.

We offer a couple of qualitative insights into the performance differences. First, KVP is contextually richer than

TABLE II
TOP-FIVE ABSTRACTION RESULTS OF ITRUST AND OPENEMR BY ABSTFINDER [4], RAI [1], AND OUR <KEY, VALUE> PAIR (KVP) APPROACH

rank	iTrust			OpenEMR		
	AbstFinder	RAI	KVP	AbstFinder	RAI	KVP
1	health	cardiac causes	<"factors", "vascular damage">	health	subsequent recovery	<"identifier type", "uniform">
2	clinical	healthcare	<"factors", "pathogens" "dysfunctions">	clinical	subsequent amendment	<"cell type", "cytokines">
3	patient	security	<"factors", "mental protection">	patient	subsequent surgical	<"type", "pharmacologic">
4	health information	natural causes	<"safety checks", "potential dose">	system	subsequent complications	<"procedure", "patient id" "name" "sex" "age">
5	electronic	reversible causes	<"x - rays", "absorbed dose">	social	ambulatory care pharmacists	<"term", "patient type">

TABLE III
COVERAGE OF THE EIGHT GISTS OF THE ENVIRONMENTAL ASSERTIONS ABOUT iTRUST'S "SCHEDULE APPOINTMENTS" USE CASE [14]

Gist	AbstFinder	RAI	KVP
valid user accounts	✓	✓	✓
valid appointment time	✓	✓	✓
valid appointment type			
unique patient account		✓	✓
valid schedule alternatives			
designated LHCP			
responding to appointment requests			✓
displaying patient message	✓	✓	✓

AbstFinder and RAI. In iTrust, the 6th candidate identified by AbstFinder is "blood", a rather general concept. RAI outputs "blood pressure" as its 45th candidate, scoping down the domain concern greatly. A relevant KVP generated by our approach is the 98th candidate: <"blood pressure", "140">, further depicting a testable and indicative domain phenomenon, i.e., 140 mm Hg or above implies Stage 2 high blood pressure.

Our second observation is that KVP tends to group relevant domain phenomena which are spread out otherwise. In OpenEMR, for instance, <"symptoms", "shortness breath" "chest pain"> is ranked 8th. RAI, on the other hand, recognizes "chest pain" as its 10th candidate and "shortness breath" as its 352nd candidate. The distance is so large that the two candidates are unlikely to be related with one another. AbstFinder is also limited in that "shortness breath" is ranked 682nd but "chest pain" is not identified at all. As shown in Fig. 8, KVP's Recall@200 reaches about 60% in both domains, covering more relevant domain phenomena than RAI and AbstFinder.

C. *Comparison with Manually Formulated Assumptions*

To gain further insights into abstractions' completeness, we compare them with manually created environmental assumptions. Our comparison is two-fold: matching the 150 iTrust assertions studied by Bhowmik *et al.* [14], and matching the 8 meeting scheduler assumptions shared by Rahimi *et al.* [20]. In both cases, two researchers manually extracted the gists of the iTrust assertions and the meeting scheduler assumptions, and then matched the automatically generated abstractions with those gists in a joint session.

Table III shows the coverage results of iTrust. As the environmental assertions in [14] are about iTrust's "Schedule Appointments" use case (cf. Fig. 2), we used only this use case's NL descriptions, instead of the NL requirements of all 54 iTrust's use cases, to rank the automatically generated abstraction candidates. Inspecting the top-200 abstractions identified by AbstFinder, RAI, and our KVP led to a 37.5%, 50%, and 62.5% coverage over the eight gists listed in Table III. For instance, "valid user account" was covered by AbstFinder's 163rd candidate ("validated patient"), RAI's 89th candidate ("valid patient"), and KVP's 59th candidate (<"user", "valid patient">). Compared to AbstFinder and RAI, KVP had the highest coverage. Notably, the 142nd KVP, <"respondents", "regular patients">, corresponded to the assertion that the respondents (e.g., a licensed health care professional) shall approve or deny the patients' requests for appointment. Such an assertion was not covered by any of the top-200 candidates generated by AbstFinder or RAI.

In a study on environmental assumptions, Rahimi and her colleagues [20] collected 150 statements from the literature (textbooks, papers, and websites). From this collection, we selected all the assumptions of the meeting scheduler domain. These eight statements, shown in the leftmost column of Table IV, were taken from van Lamsweerde [12]. The rest of Table IV provides the top-ranked, relevant KVP identified by our approach that matched the assumption. Although some statements are idiosyncratic, such as "Saturdays are excluded dates for meetings", no KVP within the top-200 abstractions was found to be relevant to the gist of not scheduling meetings on some special date. Overall, the coverage of the eight meeting scheduler assumptions [12] is 58.3% ($\frac{14}{24}$). A noteworthy finding is that, even for the same assumption, different abstractions are identified for different software products, enabling more specialized testing for each system, e.g., checking if a participant who is excluded from the "access control list" in the "OneDrive folder" could join a Teams meeting, or "ip address" should authenticate "host identification" in Zoom.

D. *Bug-Revealing Capability of Abstractions*

As we discussed in equation (4), revealing bugs shows the practical value of the abstractions in the RBT process. Among our six subject systems, we focused on the known bugs of iTrust, Teams, Webex, and Zoom. Bhowmik *et al.* [14] highlighted two defects of iTrust discovered manually in the RBT process. In our analysis, both bugs could be revealed with support of the KVP results. We manually judged which abstractions, if known to the testers, could help detect the bugs. We found that the 139th pair, <"appointment", "last year">, could help uncover the bug that iTrust allowed an appointment to be made for a past time, and the 79th pair, <"time", "scheduling conflicts"> could help detect the bug that iTrust allowed a patient to schedule appointments with multiple doctors for different reasons at the same time.

Our manual web search found 7 bugs for Teams, 5 bugs for Webex, and 7 bugs for Zoom, all of which are shared in our experimental materials. Analyzing the top-200 KVPs manually, we were able to use the abstractions to help reveal 3 Teams's bugs (43%), 2 Webex's bugs (40%), and 2 Zoom's bugs (29%). For example, Teams's 162nd pair was <"meeting organizer", "repetition occurrence meeting">, helping to define a path of RBT as follows:

1) Scheduler organizes a daily meeting series;
2) Scheduler invites a guest to join only on day #3;
3) Guest accesses the meeting series's text chats on day #4.

If the assumption at step 2) is *one-time* invitation only, then step 3) is expected to fail. However, step 3) was successful in Teams, because Teams assumed the guest invitation was from day #3 *onward*. Constructing the above testing path also needs the deep understanding of inviting a guest to a "repetition occurrence meeting". This emphasizes that the automatically identified abstractions are supporting the human

TABLE IV
TOP-RANKED <KEY, VALUE> PAIR MATCHING MEETING SCHEDULER'S ENVIRONMENTAL ASSUMPTIONS ("−−" MEANS NO RELEVANT PAIR WAS FOUND WITHIN THE TOP-200 ABSTRACTION CANDIDATES TO MATCH THE ASSUMPTION)

Environmental Assumption	Teams	Webex	Zoom
A participant cannot attend multiple meetings at the same time.	67th: <"offline attacks", "multiple user accounts">	29th: <"shared password, "another person citation">	59th: <"addition users", "passwords bookmarks" "history" "cookies">
Participants will promptly respond to e-mail requests.	58th: <"e-mail", "Microsoft 365 Business Basic">	−−	−−
A participant is on the invitee list if & only if he or she is invited to that meeting.	−−	−−	188th: <"user", "required path">
A meeting is scheduled if & only if its time and location are set.	5th: <"meetings", "Microsoft Teams">	133rd: <"meeting", "time meeting">	15th: <"meetings", "up to 100 devices" "40-minute time restriction">
A meeting is scheduled only if it is requested.	−−	−−	−−
Saturdays are excluded dates for meetings.	−−	−−	−−
Confidentiality rules can prevent non-privileged participants being aware of constraint.	181st: <"messenger rooms", "limit participant of 50">	155th: <"server hosts", "private sharing">	153rd: <"security", "unauthorized person">
Confidentiality rules can prevent non-privileged participants being aware of meetings.	173rd: <"access control list", "OneDrive folder">	165th: <"person meeting", "private meeting">	129th: <"ip address", "host identification">

analysts, rather than replacing them, in performing RBT. Nevertheless, our 65th KVP, <"sharing feature", "video sharing" "audio sharing" "desktop sharing" "file sharing" "whiteboard sharing" "text sharing">, clearly suggests some new testing paths similar to the above, where step 3) can concentrate on guest's access to day #4's uploaded files or day #4's shared whiteboard. In fact, Teams provides separate entry points to chats, files, and whiteboards. Resolving guest's chat access does *not* resolve the file or whiteboard access. Thus, related KVPs can improve the efficiency of uncovering related bugs.

E. Threats to Validity

A threat to construct validity is that our answer set's building adopted shortest-common-supersequence [53] in order to unify the abstractions identified by different techniques. This caused the matching between abstraction candidates and answer set elements to be judged on a *'contained in'* basis, which must be taken into account when interpreting our reported recall and precision values. Another threat is that our assessment of the bug-revealing capability of abstractions is only speculative in this work.

We believe the internal validity is high in that the factors potentially affecting the abstractions' accuracy, coverage, and bug-revealingness measures are under our direct control. This makes the abstraction identification techniques the cause of observed differences. One factor worth noting is that we ran our Python-based preprocessing steps in February 2021 in order to collect the 250 Wikipedia pages for each domain; however, the collected pages affected all three techniques. The comparison results among AbstFinder, RAI, and our KVP approach therefore remain valid.

Our evaluation results may not generalize to other subject systems or other domains, a threat to external validity. Studying more software applications within and beyond Electronic health records and Web conferencing will be valuable. Another threat here is our reliance on Wikipedia for abstraction identification. From a computational linguistic point of view, Wikipedia provides a balance in size, quality, and structure, between the highly-structured, but limited in coverage, linguistic databases like WordNet, and the large-scale, but less-structured, corpora such as the entire Web [31]. Nevertheless, using other corpora or combining Wikipedia with additional NLP support like WordNet could be interesting directions to expand our work.

VI. CONCLUSIONS

Automatically finding abstractions that are of particular significance in a given domain has attracted much attention in RE, though the primary focus has been on supporting early-RE activities such as requirements elicitation and modeling [1]–[4]. In this paper, we have presented an automated approach built on five novel NLP patterns to identifying abstractions in the form of <key, value> pairs. We regard such a form to be testable, and also select relevant Wikipedia pages as an indicative corpus. Evaluating our approach with six software applications in two different domains shows that the <key, value> pairs are more accurate than the abstraction candidates generated by contemporary techniques. Initial findings also indicate our abstractions' capabilities of revealing bugs and matching the environmental assumptions created manually.

Our work can be extended towards several avenues. Empirical studies, including theoretical replications [54], [55], with more software systems and more application domains are needed to lend strength to the findings reported here. Moreover, grouping related abstractions can be explored for uncovering more bugs. Finally, as the RBT advocates requirements coverage [27], opportunities exist to re-rank the abstractions by considering which requirements would be tested and which ones would need to be tested.

ACKNOWLEDGMENT

The research is partially supported by the National Natural Science Foundation of China under Grant No. 61620106007, 61802374, 62002348, and 62072442, the National Key Research and Development Program of China under Grant No. 2018YFB1403400, and Youth Innovation Promotion Association CAS.

References

[1] R. Gacitua, P. Sawyer, and V. Gervasi, “Relevance-based abstraction identification: technique and evaluation,” *Requirements Engineering*, vol. 16, no. 3, pp. 251–265, September 2011.

[2] P. Sawyer, P. Rayson, and K. Cosh, “Shallow knowledge as an aid to deep understanding in early phase requirements engineering,” *IEEE Transactions on Software Engineering*, vol. 31, no. 11, pp. 969–981, November 2005.

[3] R. Gacitua, P. Sawyer, and V. Gervasi, “On the effectiveness of abstraction identification in requirements engineering,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Sydney, Australia, September-October 2010, pp. 5–14.

[4] L. Goldin and D. M. Berry, “AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation,” *Automated Software Engineering*, vol. 4, no. 4, pp. 375–412, October 1997.

[5] A. Dwarakanath, R. R. Ramnani, and S. Sengupta, “Automatic extraction of glossary terms from natural language requirements,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Rio de Janeiro, Brazil, July 2013, pp. 314–319.

[6] C. Arora, M. Sabetzadeh, L. C. Briand, and F. Zimmer, “Automated extraction and clustering of requirements glossary terms,” *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 918–945, October 2017.

[7] T. Gemkow, M. Conzelmann, K. Hartig, and A. Vogelsang, “Automatic glossary term extraction from large-scale requirements specifications,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Banff, Canada, August 2018, pp. 412–417.

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

[9] K. Ryan, “The role of natural language in requirements engineering,” in *Proceedings of the International Symposium on Requirements Engineering (RE)*, San Diego, CA, USA, January 1993, pp. 240–242.

[10] Z. Jin, *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*. Morgan Kaufmann, 2018.

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

[12] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[13] T. T. Tun, R. R. Lutz, B. Nakayama, Y. Yu, D. Mathur, and B. Nuseibeh, “The role of environmental assumptions in failures of DNA nanosystems,” in *Proceedings of the International Workshop on Complex Faults and Failures in Large Software Systems (COUFLESS)*, Florence, Italy, May 2015, pp. 27–33.

[14] T. Bhowmik, S. R. Chekuri, A. Q. Do, W. Wang, and N. Niu, “The role of environment assertions in requirements-based testing,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, September 2019, pp. 75–85.

[15] N. Niu and A. Mahmoud, “Enhancing candidate link generation for requirements tracing: the cluster hypothesis revisited,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Chicago, IL, USA, September 2012, pp. 81–90.

[16] W. Wang, A. Gupta, N. Niu, L. D. Xu, J.-R. C. Cheng, and Z. Niu, “Automatically tracing dependability requirements via term-based relevance feedback,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 342–349, January 2018.

[17] W. Wang, N. Niu, H. Liu, and Z. Niu, “Enhancing automated requirements traceability by resolving polysemy,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Banff, Canada, August 2018, pp. 40–51.

[18] J. Wermter and U. Hahn, “Finding new terminology in very large corpora,” in *Proceedings of the International Conference on Knowledge Capture (K-CAP)*, Banff, Canada, October 2005, pp. 137–144.

[19] J. C. Knight, “Safety critical systems: challenges and directions,” in *Proceedings of International Conference on Software Engineering (ICSE)*, Orlando, Florida, USA, May 2002, pp. 547–550.

[20] M. Rahimi, W. Xiong, J. Cleland-Huang, and R. R. Lutz, “Diagnosing assumption problems in safety-critical products,” in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, Urbana, IL, USA, October-November 2017, pp. 473–484.

[21] M. Alenazi, N. Niu, and J. Savolainen, “A novel approach to tracing safety requirements and state-based design models,” in *Proceedings of International Conference on Software Engineering (ICSE)*, Seoul, South Korea, June-July 2020, pp. 848–860.

[22] C. Yang, P. Liang, and P. Avgeriou, “Assumptions and their management in software development: a systematic mapping study,” *Information & Software Technology*, vol. 94, pp. 82–110, February 2018.

[23] X. Jin, C. Khatwani, N. Niu, M. Wagner, and J. Savolainen, “Pragmatic software reuse in bioinformatics: how can social network information help?” in *Proceedings of International Conference on Software Reuse (ICSR)*, Limassol, Cyprus, June 2016, pp. 247–264.

[24] J. Bhuta and B. Boehm, “A framework for identification and resolution of interoperability mismatches in COTS-based systems,” in *Proceedings of the International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques (IWICSS)*, Minneapolis, MN, USA, May 2007.

[25] W. Wang, F. Dumont, N. Niu, and G. Horton, “Detecting software security vulnerabilities via requirements dependency analysis,” *IEEE Transactions on Software Engineering*, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2020.3030745

[26] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner’s Guide to the RUP*. Addison-Wesley, 2003.

[27] P. Skoković and M. Rakić-Skoković, “Requirements-based testing process in practice,” *International Journal of Industrial Engineering and Management*, vol. 1, no. 4, pp. 155–161, 2010.

[28] A. Meneely, B. Smith, and L. Williams, “iTrust electronic health care system: a case study,” in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012.

[29] 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.

[30] C. A. Gunter, E. L. Gunter, M. Jackson, and P. Zave, “A reference model for requirements and specifications,” *IEEE Software*, vol. 17, no. 3, pp. 37–43, May/June 2000.

[31] A. Mahmoud and N. Niu, “On the role of semantics in automated requirements tracing,” *Requirements Engineering*, vol. 20, no. 3, pp. 281–300, September 2015.

[32] S. Ezzini, S. Abualhaija, C. Arora, M. Sabetzadeh, and L. C. Briand, “Using domain-specific corpora for improved handling of ambiguity in requirements,” in *Proceedings of International Conference on Software Engineering (ICSE)*, Madrid, Spain, May 2021, pp. 1485–1497.

[33] S. Chernov, T. Iofciu, W. Nejdl, and X. Zhou, “Extracting semantics relationships between wikipedia categories,” in *Proceedings of the Workshop on Semantic Wikis (SemWiki)*, Budva, Montenegro, June 2006.

[34] Beautiful Soup, “A Python Library for Pulling Data out of HTML and XML Files,” Last accessed: July 2021. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/

[35] S. Abualhaija, C. Arora, M. Sabetzadeh, L. C. Briand, and E. Vaz, “A machine learning-based approach for demarcating requirements in textual specifications,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, September 2019, pp. 51–62.

[36] spaCy, “Industrial-Strength Natural Language Processing In Python,” Last accessed: July 2021. [Online]. Available: https://spacy.io/

[37] S. Kübler, R. McDonald, and J. Nivre, *Dependency Parsing*. Morgan & Claypool Publishers, 2009.

[38] F. Dalpiaz, D. Dell’Anna, F. B. Aydemir, and S. Çevikol, “Requirements classification with interpretable machine learning and dependency parsing,” in *Proceedings of the International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, September 2019, pp. 142–152.

[39] K. B. Cohen, T. Christiansen, and L. E. Hunter, “Parenthetically speaking: classifying the contents of parentheses for text mining,” in *Proceedings of the Annual Symposium on Biomedical and Health Informatics (AMIA)*, Washington, DC, USA, October 2011, pp. 267–272.

[40] C. Klaussner and D. Zhekova, “Pattern-based ontology construction from selected Wikipedia pages,” in *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP) Student Research Workshop*, Hissar, Bulgaria, September 2011, pp. 103–108.

[41] M. A. Hearst, “Automatic acquisition of hyponyms from large text corpora,” in *Proceedings of the International Conference on Computational Linguistics (COLING)*, Nantes, France, August 1992, pp. 539–545.

[42] C. Klaussner and D. Zhekova, "Lexico-syntactic patterns for automatic ontology building," in *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP) Student Research Workshop*, Hissar, Bulgaria, September 2011, pp. 109–114.

[43] T. W. Finin, "The semantic interpretation of nominal compounds," in *Proceedings of the Annual National Conference on Artificial Intelligence (AAAI)*, Stanford, CA, USA, August 1980, pp. 310–312.

[44] H. Liu, M. Shen, J. Zhu, N. Niu, G. Li, and L. Zhang, "Deep learning based program generation from requirements text: Are we there yet?" *IEEE Transactions on Software Engineering*, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2020.3018481

[45] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer, and Z. Niu, "Automated recommendation of software refactorings based on feature requests," in *Proceedings of the International Requirements Engineering Conference (RE)*, Jeju Island, South Korea, September 2019, pp. 187–198.

[46] A. S. Nyamawe, H. Liu, Z. Niu, W. Wang, and N. Niu, "Recommending refactoring solutions based on traceability and code metrics," *IEEE Access*, vol. 6, pp. 49 460–49 475, 2018.

[47] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer, and Z. Niu, "Feature requests-based recommendation of software refactorings," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4315–4347, September 2020.

[48] N. Niu, J. Savolainen, T. Bhowmik, A. Mahmoud, and S. Reddivari, "A framework for examining topical locality in object-oriented software," in *Proceedings of the Annual IEEE Computer Software and Applications Conference (COMPSAC)*, Izmir, Turkey, July 2012, pp. 219–224.

[49] OpenEMR, "A Medical Practice Management Software System Supporting Electronic Medical Records (EMR)," Last accessed: July 2021. [Online]. Available: https://en.wikipedia.org/wiki/OpenEMR

[50] OpenMRS, "A Collaborative Open-Source Project on Medical Record Systems (MRS)," Last accessed: July 2021. [Online]. Available: https://en.wikipedia.org/wiki/OpenMRS

[51] OpenEMR Features, "Features of OpenEMR," Last accessed: July 2021. [Online]. Available: https://www.open-emr.org/wiki/index.php/OpenEMR_Features

[52] OpenMRS User Guide, "A Complete User Guide for OpenMRS," Last accessed: July 2021. [Online]. Available: https://wiki.openmrs.org/display/docs/User+Guide

[53] D. Maier, "The complexity of some problems on subsequences and supersequences," *Journal of the ACM*, vol. 25, no. 2, pp. 322–336, September 1978.

[54] 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 *Proceedings of the International Requirements Engineering Conference (RE)*, Beijing, China, September 2016, pp. 186–195.

[55] 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.