The Journal of Systems & Software 170 (2020) 110730

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Capturing creative requirements via requirements reuse: A machine learning-based approach

Quoc Anh Do[a,*], Tanmay Bhowmik[a], Gary L. Bradshaw[b]

[a] Department of Computer Science and Engineering, Mississippi State University, Mississippi State, MS, 39762, USA
[b] Department of Psychology, Mississippi State University, Mississippi State, MS, 39762, USA

ARTICLE INFO

Article history:
Received 25 October 2019
Received in revised form 28 June 2020
Accepted 7 July 2020
Available online 9 July 2020

Keywords:
Requirements reuse
Requirements engineering
Creativity in RE
Boilerplate
Natural language processing
Machine learning

ABSTRACT

The software industry has become increasingly competitive as we see multiple software serving the same domain and striving for customers. To that end, modern software needs to provide creative features to improve sustainability. To advance software creativity, research has proposed several techniques, including multi-day workshops involving experienced requirements analysts, and semi-automated tools to support creative thinking in a limited scope. Such approaches are either useful only for software with already rich issue tracking systems, or require substantial engagement from analysts with creative minds. In a recent work, we have demonstrated a novel framework that is beneficial for both novel and existing software and allows end-to-end automation promoting creativity. The framework reuses requirements from similar software freely available online, utilizes advanced natural language processing and machine learning techniques, and leverages the concept of requirement boilerplate to generate candidate creative requirements. An application of our framework on software domains: Antivirus, Web Browser, and File Sharing followed by a human subject evaluation have shown promising results. In this invited extension, we present further analysis for our research questions and report an additional evaluation by human subjects. The results exhibit the framework's ability in generating creative features even for a relatively matured application domain, such as Web Browser, and provoking creative thinking among developers irrespective of their experience levels.

1. Introduction

Software requirements describe, often in natural language, the services expected of a software system addressing the needs and desires of its stakeholders (Nuseibeh and Easterbrook, 2000). Requirements engineering (RE) is the process that involves identification and documentation of such requirements in order to facilitate further analysis, communication, and implementation activities (Nuseibeh and Easterbrook, 2000). Traditional RE, for the most part, builds upon the belief that requirements remain in the stakeholders' minds in an implicit manner (Lemos et al., 2012), and focuses on techniques and models that help elicitation and documentation of those requirements. In modern internet era, however, we observe an accelerated growth in the software market, where the same application domain is served by multiple software systems that are competing for end-users. As a result, this aggressive modern market naturally favors software which provides novel and useful features (Maiden et al., 2010). Consequently, contemporary requirements engineers must capture innovative requirements so that their software can be equipped with competitive advantage. Therefore, highlighting the key role RE needs to play in improving a software system's sustainability by aiding the capture of more useful and novel requirements, researchers have recently framed RE as a creative problem solving process (Maiden et al., 2010).

Creativity, considered as a multidisciplinary area of research, is commonly known as "the ability to produce work that is both *novel* (i.e., original and unexpected) and *appropriate* (i.e., useful and adaptive to the task constraints)" (Sternberg and Sternberg, 1999). Creativity in RE, as Maiden et al. defines, is the capture of requirements new to the project stakeholders but may not be historically new to humankind (Maiden et al., 2010). Investigations into creativity in other disciplines, including psychology, indicate that we can achieve creativity in RE, through *exploration*, *combination*, and *transformation* of ideas that exist in the conceptual domain (Maiden et al., 2010; Boden, 2003), commonly known as exploratory, combinational, and transformational creativity, respectively (Maiden et al., 2010). The most common approach to promote creativity in RE involves multi-day workshops

* Corresponding author.
E-mail addresses: aqd14@msstate.edu (Q.A. Do), tbhowmik@cse.msstate.edu (T. Bhowmik), glb2@psychology.msstate.edu (G.L. Bradshaw).

https://doi.org/10.1016/j.jss.2020.110730
0164-1212/© 2020 Elsevier Inc. All rights reserved.

with experienced human facilitators where ideas for requirements need to be generated through manual activities (Maiden et al., 2004a,b). Although such an approach is found to be successful in supporting software creativity, associated costs due to economic, time and geographical pressures preclude a wide adoption of this intensive processes (Horkoff and Maiden, 2015). Research has also investigated frameworks (Bhowmik et al., 2014, 2015; Do and Bhowmik, 2018), tools (Zachos and Maiden, 2008), and techniques (Burnay et al., 2016; Murukannaiah et al., 2016) that are semi-automated in nature in order to support creative thinking. However, these approaches are still time consuming (Murukannaiah et al., 2016), requires significant involvement from human analysts who have creative abilities (Burnay et al., 2016; Do and Bhowmik, 2018; Zachos and Maiden, 2008), and are applicable only for existing software for which we have rich and detailed issue tracking systems (Bhowmik et al., 2014, 2015).

This work consists of two closely related studies. In the first study, we propose a framework that reuses software requirements, freely available online, for other software in the application domain and generates candidate creative requirements in an automated manner (Do et al., 2019). The study also validates the effectiveness of our proposed approach by conducting an initial human subject evaluation with 30 participants. The results indicate that our framework can exhibit promising performance in promoting creativity by inspiring creative thinking among developers irrespective of their experience levels. Further qualitative analysis on the collected data has also uncovered certain interesting questions about our framework. In the second study, we present additional statistical analysis for our earlier research questions and conduct a follow-up study with domain experts. The results suggest our framework's ability to promote creativity even for a relatively matured software domain and to further boost innovative aspects through refined requirements originated from the automated ones. In addition, the second study offers some implications of our findings for practitioners, in particular, guidelines on how practitioners and stakeholders can utilize and be benefited from our framework.

Through this work, we contribute a framework that automatically generates concise and possibly creative requirements utilizing modern NLP and ML techniques. In addition, our framework can promote creativity in RE, for both novel and existing software, requiring limited human interventions compared to the known approaches. In what follows, Section 2 presents some important work related to our research. Section 3 provides background information about some of the techniques we have used in our work. Section 4.1 presents an introduction and demonstration of our framework. Section 4.2 details the initial human subject study, which is followed by Section 4.3 presenting further discussion and uncovering some intriguing questions about our framework for additional investigations. Section 5 presents the design of our follow-up study involving domain experts followed by statistical analyses and discussion of results. Some implications of the findings from our follow-up study is highlighted in Section 6. Potential validity threats of our work are discussed in Section 7. Section 8 provides some concluding remarks followed by Section 9 highlighting several directions towards our future work.

2. Related work

In this section, we discuss several related work on creativity, its evaluation, and creativity in RE, especially the manual and automated approaches we find in the literature that support creativity triggers or new ideas for requirements. In addition, we present a comparison of our work with this existing literature. It should be noted that some research we discuss here, in particular, the automated approaches, commonly leverage NLP techniques which we do not elaborate separately in this section.

2.1. *Creativity: As we find in the literature*

Creativity has been widely studied in Psychology where traditional researchers, for the most part, have regarded it as a human attribute. Recently, however, researchers have converged on a *product*-oriented characterization of the notion of creativity. In this section, we first present the traditional approach of defining creativity and further elaborate on the product-oriented definition.

2.1.1. *Defining creativity*

Although creativity have been studied extensively since 1950s in various human-centric disciplines, including psychology, economics, product management, and innovation (Sternberg and Sternberg, 1999), the early research era lacked conscientious on the definition or operationalization of creativity (Corazza, 2016; Ford and Harris, 1992; Parkhurst, 1999; Plucker et al., 2004; Runco and Jaeger, 2012). Nevertheless, many creativity researchers gradually reached the *standard definition* of creativity, derived from the pioneering works of Barron (1955) and Stein (1953), and put emphasis on some of creativity's key attributes. For example, Sternberg defines creativity as "the *ability* to produce work that is both *novel* (i.e., original, expected) and *appropriate* (i.e., useful, adaptive concerning the task constraints)" (Sternberg and Sternberg, 1999). In recent years, however, the *product*-oriented approach of defining creativity, i.e., the novelty and usefulness of the product, has become dominant as Mumford points out (Mumford, 2003), "Over the course of the last decade, however, we seem to have reached a general agreement that creativity involves the production of novel, useful products". Along this line, Runco and Jaeger (2012) further offer an in-depth look at the originality (i.e., novelty) and usefulness aspects and state that "originality is vital for creativity but is not sufficient as we should not neglect the importance of *usefulness*". In our work, we follow the product-oriented approach and adapt the novel and useful *requirement*-oriented approach of defining creativity in RE.

2.1.2. *Measuring creativity*

When it comes to measuring creativity, several important, yet challenging, questions such as "how much novelty?", "how much usefulness?", "useful to whom?", or "who would judge the novelty and usefulness?" remain unclear, thereby making measuring creativity exceedingly difficult. Despite such challenges in terms of creativity assessment or measurement, there have been attempts to tackle the task focusing on defining and measuring key attributes of *product*-creativity. O'Quin and Besemer (1989) used the Creative Product Semantic Scale (CPSS), which assesses product based on three factors: novelty, resolution, and elaboration & Synthesis. They used CPSS to evaluate 3 products from a department store catalog (O'Quin and Besemer, 1989). CPSS was also used to evaluate the creativity of print advertisements by three groups of participants with diverse backgrounds, including college students, advertising professionals, and the general public (White et al., 2002). In an effort to enhance traditional creativity assessment approaches, Cropley and Cropley (2005) proposed a 27-item Creative Solution Diagnosis Scale (CSDS) advocating Likert scale ratings along four different dimensions: relevance & effectiveness, novelty, elegance, and genesis. The idea is that a beholder can look into a product from the perspective of each of the 27 indicators and provide separate ratings, thereby evaluating its creative merit. A major drawback of this scale is that, given the large number of indicators, evaluating even a few products will be very tedious and time consuming. Such an issue hinders a wide adoption of CSDS as a practical measurement scale for product-creativity. In our work, without losing the essence of creativity definition, we employ three straightforward criteria, i.e., *clarity*, *novelty*, and *usefulness*, to measure the creativity aspects of software requirements.

2.2. Creativity in RE

Maiden et al. (2010) have recently framed RE as a creative process where stakeholders, including but not limited to requirements engineers, software developers, and team leaders, closely work together to come up with innovative ideas for software features. Research on creativity in RE is often associated with providing creativity triggers for the stakeholders so that they can develop innovative ideas leveraging those triggers. We present the related work on creativity in RE by broadly classifying existing research into two categories: *manual* and *automated* approaches to formulate creative requirements.

Manual Approaches. Traditionally, the process of developing creativity triggers often involved techniques such as creativity workshops that require commitment of substantial time and intellectual labor from multiple human facilitators. To that end, most of the earlier literature in creativity in RE predominantly focused on workshops designed to facilitate creative thinking. These workshops typically spanned over multiple days with intense involvement from experienced human facilitators. In one of such earlier works, for example, Maiden et al. (2004a) organized workshops where the stakeholders were particularly encouraged to brainstorm and think creatively during a requirements process. In order to capture system requirements, Maiden and colleagues (Maiden et al., 2004b) also proposed RESCUE, which is a scenario-based RE process that involves creativity workshops. A limitation of creativity techniques involving such workshops is that they require an end-to-end engagement from human facilitators (Horkoff and Maiden, 2015) and heavily depend on the "creative muse" of the participants.

Recent research has proposed some tools and frameworks to support creative thinking while capturing requirements. As an example, we can mention the algorithmic solution developed by Zachos and Maiden (2008) that explored web services in the domain analogous to current requirement problem. With an aim to identify requirements from scenarios and to promote creative thinking while collecting information for requirements, an integrated software tool was proposed by Karlsen et al. (2009). Sakhnini and colleagues (Sakhnini et al., 2012) applied the elementary pragmatic model, also known as EPM (Lefons et al., 1977), to support elicitation of creative requirements. In a recent work, Burnay et al. (2016) experimented with creativity triggers as a technique to stimulate stakeholders' imagination. Murukannaiah and colleagues (Murukannaiah et al., 2016) proposed a sequential process through which potentially creative requirement could be manually acquired from the crowd (Groen et al., 2015). Sakhnini et al. (2012) also introduced a creativity enhancement technique named Power-Only EPMcreate, which aims to expand creativity search space by engaging two stakeholders in a series of logical combination of their viewpoints. The tools and frameworks mentioned so far mostly provide a partial support and still require considerable human involvement to generate new ideas. In contrast, our approach offers a fully-automated solution for those who are looking for next requirements ideas. Currently, it supports generating ideas for three popular domains, which are Antivirus, Web Browser, and File Sharing, but could be readily scaled to include more as needs arise.

Automated Creation of Software Requirements. Automated generation of creative software requirements has received a limited attention so far. Existing literature provides a few examples that involve some level of automation to a certain degree. For instance, Farfeleder and colleagues (Farfeleder et al., 2011) developed an automated semantic guidance system leveraging domain ontology and supported the formulation of new requirements building upon manually defined attributes, such as concepts, relations, and axioms. Some researchers have also focused on the automated creation of requirements triggers that can aid elicitation of creative requirements (Sakhnini et al., 2010; El-Sharkawy and Schmid, 2011). To that end, Bhowmik et al. (2014) presented a framework providing an automated support to create new requirements applying topic modeling (Linstead et al., 2008) and part-of-speech tagging (Brill, 1992) on the existing requirements of the *same* software system. A limitation of this framework is that it is only applicable for a software that has a long history over issue tracking system. Furthermore, it needs to collect and handle more complex data that include requirements, identity of specific stakeholders, and their contributions to the issue tracking system in terms of posted comments and artifacts. Another effort for automation was recently reported by Do and Bhowmik (2018) where the researchers experimented with Hidden Markov Model (HMM) to automatically generate requirements based on hidden creative attributes of existing requirements. The outcomes of this approach, however, are extremely random as the generated statements are sequence of apparently unrelated words providing very limited understandability. El-Sharkawy and Schimid implemented a semantic-based heuristic approach examining relations among existing requirements to derive new idea triggers (El-Sharkawy and Schmid, 2011). In addition to representing ideas or creativity triggers as a directed graph, El-Sharkawy and Schmid (2011) presented six production rules to systematically retrieve new innovative ideas based on derived associations among existing ideas.

The aforementioned works have helped us to enhance automated approach to address noticeable gaps and to offer some practical improvements aiming to (1) significantly reduce human effort and involvement on generating creative requirements and (2) seamlessly support multiple approaches to accomplish creativity, including exploratory and combinational creativity techniques (Maiden et al., 2010). To that end, in (Do et al., 2019), we have developed an automated framework and applied it on Antivirus, Web Browser, and File Sharing, three popular application domains found on Softpedia. Do et al. (2019) has also presented an initial human subject evaluation of our framework with 30 participants. The results from this evaluation indicate that our framework can exhibit promising performance in promoting creativity by inspiring creative thinking among developers irrespective of their experience levels. Further qualitative analysis on the collected data has also uncovered certain interesting questions about our framework. Motivated by such findings, in this invited extension, we present additional statistical analysis for our earlier research questions, conduct a follow-up study with domain experts, and further discuss implications of our findings for practitioners. Overall, we demonstrate that our framework is useful for both new and existing software systems. It also generates requirements with improved meanings compared to the HMM-based approach proposed by Do and Bhowmik (2018). Our approach is also highly flexible in the sense it can be quickly extended to support combinational creativity (Maiden et al., 2010) from a cross-domain perspective. The techniques we utilized in this paper, in particular, the word embedding method doc2vec (Le and Mikolov, 2014) and clustering algorithm BIRCH (Zhang et al., 1996), are greatly effective and efficient in terms of processing large multi-dimensional dataset which would result in a scalable solution for requirements generation.

Evaluating Creativity in RE. Following other disciplines, the RE research community has recently adopted the "new and useful" definition of creativity (Maiden et al., 2010; Burnay et al., 2016; Murukannaiah et al., 2016; Bhowmik et al., 2015) and used various evaluation schemes. While capturing features for an air space management system, Maiden et al. (2004a,b) and Maiden and Robertson (2005) followed a simple approach to measure "novel" and "appropriate" separately on a 3-point Likert scale (Maiden et al., 2007) and collected ratings from two domain experts. In the case of a security access system, Karlsen et al. (2009) asked an expert to rate each requirement at a binary scale (i.e., creative or not creative). While capturing requirements from web services, Zachos and Maiden (2008) equated *novelty* to textual *dissimilarity* to assess creativity. In Bhowmik et al. (2014) and Bhowmik et al. (2015), Bhowmik et al. relied on a single Likert scale rating for "novelty and usefulness". Bhowmik et al. (2015) also included an informal *self-rating*, i.e., scores from individuals who participated in the creation process. Burnay et al. (2016) recently used feedback from eight students (non-experts) in order to assess the effectiveness of six new creativity triggers (Burnay et al., 2016). Murukannaiah et al. (2016) collected separate ratings for clarity, novelty, and usefulness from three hundred participants on a 5-point Likert scale. In our work, we follow the product-oriented approach and adapt the novel and useful *requirement*-oriented definition of creativity in RE. For evaluation, we follow the approach used by Murukannaiah et al. (2016) which facilitates novelty, usefulness, as well as clarity.

3. Background

In order to make our framework more generalizable, we leverage certain key techniques that are explained in this section. Hereafter, this manuscript considers *requirements analysts*, indicating stakeholders who perform elicitation and elaboration of software requirements, and *requirements engineers* as synonymous. In addition, *requirement* and *feature* are two terms used interchangeably.

3.1. An overview on requirement boilerplate

Since the dawn of RE, natural language (NL) has been the dominant medium for the documentation and communication of software requirements. As the nature of NL is inherently informal, it often leads to problems such as ambiguity and avoidable complexity. To mitigate this issue, the RE community has suggested boilerplates (Pohl, 2010), i.e., reusable templates that contain certain placeholders which we can replace with relevant attributes, thereby formulating consistently-structured requirement statements. Since, a major goal of this work is to generate such statements, boilerplates play a critical part in our approach. Rupp's boilerplate (Pohl, 2010) and EARS boilerplate (Mavin et al., 2009) are two commonly referenced boilerplates we find in the RE literature. Considering EARS, however, Rupp's boilerplate is known as more compact but highly effective. In order to maintain simplicity, this work adopts the following abridged version of Rupp's boilerplate (Pohl, 2010).

Rupp's boilerplate.

<**system name**>shall provide *user* with the ability to <**process**> <**object**>[**additional details about object**]

Example.

$\overbrace{\textbf{Antivirus system}}^{\text{system}}$ shall provide user with the ability to $\overbrace{\textbf{fast scan}}^{\text{process}}$ $\overbrace{\textbf{files}}^{\text{object}}$

depending on $\overbrace{\textbf{the computer specification}}^{\text{additional details}}$

As the example presented above indicates, in this work we concentrate on obtaining 3-tuples (process, object, additional details) of associated words and phrases in order to complete the boilerplate (Pohl, 2010). Here, the aim is to enhance the meaning of generated requirements. As defined by Pohl (2010), the definitions of the items in the 3-tuple are as follows.

Process: A verb or verb phrase describing an action or functionality users can use to fulfill a certain task.

Object: A noun or noun phrase describing an item the process acts upon.

Additional details: A noun phrase containing more context for the feature.

In this work, we deploy certain NLP techniques and extract the items in the 3-tuples from requirements collected from freely available online resources (in case of this paper, Softpedia). Next, we provide an overview on those NLP techniques.

3.2. An overview of the NLP techniques we leverage

In order to extract important information from software artifacts, such as bug reports, source code, and requirement documents, some NLP techniques have been extensively applied by the RE community (Bhowmik et al., 2014; Arora et al., 2016). Two of those techniques, namely Part-of-Speech (POS) Tagging and Text Chunking, play a pivotal role in our proposed framework. In what follows, we provide a brief overview on these two techniques.

POS Tagging. It is a process commonly used to assign appropriate part of speech to individual words in a string called sentence. An effective POS tagger needs to be trained on a very large corpus, usually with at least a few million words (e.g., the Penn Tree Bank dataset Marcus et al., 1993). As an example, Fig. 1 presents a tagged sentence representing a requirement. The POS tag for a word is typically determined considering its neighbors and surrounding contexts. POS tagging arguably acts as a foundational block for further advanced NLP techniques.

Text Chunking. It is a procedure to split the text into chunks (Berwick et al., 1991). Here, a chunk is defined as a non-overlapping segment of words that are syntactically related. With the notion of chunk, we can identify important ideas and keywords in a long text. Some common examples of text chunks include prepositional phrase (PP), verb phrase (VP), and noun phrase (NP). A sample requirement annotated with text chunks is presented in Fig. 1. In this example, *antivirus system*, formed by two different nouns: antivirus and system, is a noun phrase.

3.3. An overview of the language model we use

In our context, the capture of 3-tuples (cf. Section 3.1) require predicting the probable words that appear next, depending on the words that precede. To that end, we need to process a large corpus of textual data. In order to make such predictions, a conventional count-based language model, such as n-gram, could be a potential option. However, a disadvantage of such a model is that, due to "the curse of dimensionality" (Bengio et al., 2003), both time and space complexities increase exponentially as the amount of data increases. Consequently, in order to make our framework capable of processing a large amount of data, we need to employ an advanced technique to extract the 3-tuples. With that objective in mind, we pick doc2vec (Le and Mikolov, 2014), which is a generalization of a popular word embedding method called word2vec (Mikolov et al., 2013). In order to capture fixed-length vector representations for variable-length text inputs, including sentences and paragraphs, Le and Mikolov (2014) developed doc2vec, which is essentially an unsupervised ML algorithm. While count-based models represent words in terms of discrete values, text documents are encoded in contentious-numbered vectors by doc2vec. This approach followed in doc2vec reduces the model's complexity, and at the same time, it allows us to identify the semantic and syntactic relationships among documents (Le and Mikolov, 2014).

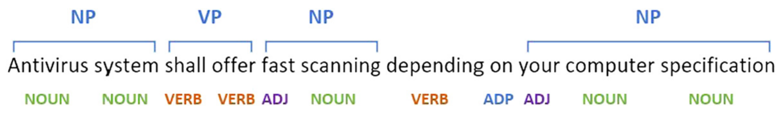

NP
VP
NP
NP
Antivirus system shall offer fast scanning depending on your computer specification
NOUN NOUN VERB VERB ADJ NOUN VERB ADP ADJ NOUN NOUN

Fig. 1. Part-of-speech and text chunking example.

4. First study

4.1. The framework

Fig. 2 presents the framework we propose to generate creative requirements via requirements reuse. Essentially, our framework utilizes and reuses existing software requirements recorded on online product listings (i.e., Softpedia) as input, and produces candidate creative requirements as output. As the figure indicates, the entities involved in our framework could be divided into two categories: (i) external entities and (ii) internal components. The external entities include resources such as online product listings on the internet and the requirements analyst who will ultimate analyze the candidate creative requirements. The internal components, on the other hand, involve "under the hood" items from requirements scraper to candidate selector (cf. Fig. 2). In particular, the framework acts in a sequential process that starts with a web scraping tool collecting requirements from online product listings. These requirements move through various steps that include processes such as filtering and clustering. Additional techniques, such as POS tagging, text chunking, and similarity analysis, are leveraged to capture necessary attributes that can be used in Rupp's boilerplate (Pohl, 2010) to formulate candidate creative requirements. At this stage, an analyst, indicated by the human icon in Fig. 2, will be able to evaluate and elaborate, if necessary, the candidate requirements according to his or her expectations or needs.[1] Table 1 provides an overview of components in our approach. In what follows in this section, each phase of our framework is discussed in detail.

4.1.1. Collecting requirements from online resources

The first step of our framework involves collecting a sufficient number of software requirements necessary for the following activities, such as clustering. In general, software firms use software requirements specification (often known by its acronym, SRS), in addition to project management tools, such as Jira,[2] to keep track of the requirements. On one hand, for a closed source software, such artifacts are typically copyrighted intellectual properties. On the other hand, open source systems often restrict access to these artifacts to active contributors only. Therefore, building a large dataset of software requirements could be a challenging task. On the bright side, however, software firms often use various software distribution platforms (SDPs), also know as software product listings, in order to promote their products to a wide range of users. Using these SDPs, in addition to providing easily accessible download/install options, developers also enlist a product's core feature descriptions (or requirements) that could be appealing to potential users. Therefore, we posit such SDPs to be good resources in our context for collecting requirements.

After considering various SDPs, we chose Softpedia,[3] an online software and technology distribution platform based in Romania. The rationale behind choosing Softpedia is twofold. First, it has a large and diverse collection of software spanning across numerous domains, therefore, assures an abundance of enlisted feature descriptions. Second, based on our investigation, the types of features and the quality of their descriptions provided on Softpedia are better aligned with our purpose compared to other SDPs we explored. For example, the core features for an application posted on Softpedia are carefully picked by developers and are written concisely excluding unnecessary advertising information, which we found prevalent in other SDPs such as Google Play[4] or Apple's App Store.[5]

We developed a requirements scraper in Python to exhaustively collect all features in certain domains from Softpedia. Given a domain, our scraper can visit all applications belonging to that domain and extract all listed features. Fig. 3 provides a snippet for TeamViewer, a remote control software listed on Softpedia, showing its core features that would be collected by our scraper.

Selection of a Software Domain. To demonstrate our framework, we collect requirements for the software systems from Antivirus, Web Browser, and File Sharing, three popular application domains in Softpedia. The rationale behind choosing these domains is threefold. First, we expect that these domains will provide us with an enriched requirement dataset as they top Softpedia's chart with most applications listed. Second, as these domains are commonly used, a large number of users may benefit from the created requirements. And finally, the participants in the human subject evaluation we plan to conduct (cf. Section 4.2) will be able to evaluate the creative merits of our formulated requirements in a critical manner.

4.1.2. Requirements filtering

In order to successfully solve data oriented problems, we need to ensure that the collected data is of high quality. An important characteristic of the requirements we collect for this study is that they come from various resources and are enlisted by contributors with a diverse background. As a result, these requirements are somewhat untidy in nature. Therefore, we must prepare a clean dataset before we can proceed with further processing. The particular activities we carry out for the purpose of data cleanup are as follows.

1. First, we eliminate the feature descriptions that have either too few or too many words. After conducting some analysis on our collected requirements, we realize that a feature description with just a few words generally present a short phrase indicating a functionality. Therefore, such short descriptions are not suitable for extracting boilerplate attributes (cf. Section 3.1). One example along this line would be Google Chrome. Out of its 17 core features listed on Softpedia, Google Chrome has 10 feature descriptions with length ranging from 2 to 4 words such as "Dynamic tabs", "Crash control", or "One box for everything" (Google, 2020). These short phrases, without being put into their contexts, are difficult to fully comprehend not to mention extracting useful information from them. Whereas, a feature description with many words either represents an advertisement of the product or typically includes many unrelated terms and technical words that may not be suitable for Rupp's boilerplate. For example,

1 The source code is available at http:tinyurl.com/y6pgqn2f.

2 https://www.atlassian.com/software/jira.

3 https://www.softpedia.com/.

4 https://play.google.com/.

5 https://www.apple.com/ios/app-store/.

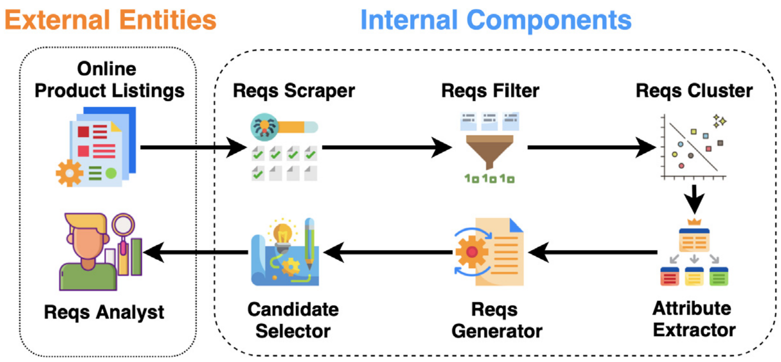
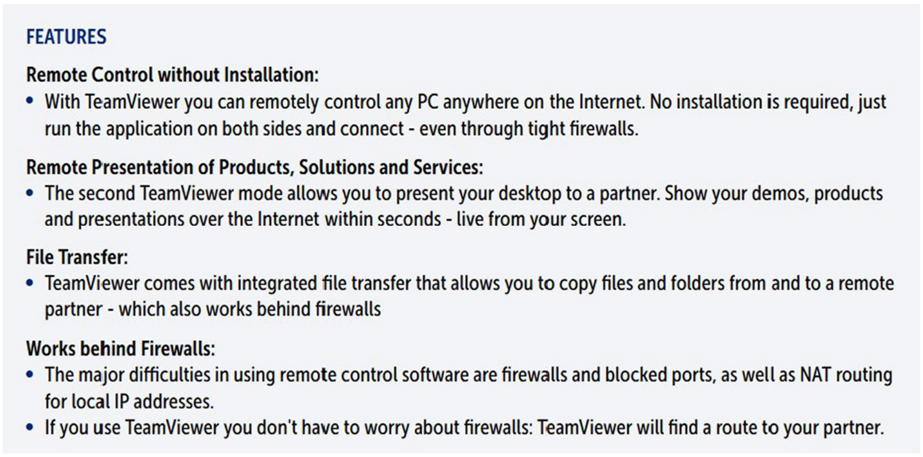


Fig. 2. A framework to capture creative requirements.



FEATURES

Remote Control without Installation:
- With TeamViewer you can remotely control any PC anywhere on the Internet. No installation is required, just run the application on both sides and connect - even through tight firewalls.

Remote Presentation of Products, Solutions and Services:
- The second TeamViewer mode allows you to present your desktop to a partner. Show your demos, products and presentations over the Internet within seconds - live from your screen.

File Transfer:
- TeamViewer comes with integrated file transfer that allows you to copy files and folders from and to a remote partner - which also works behind firewalls

Works behind Firewalls:
- The major difficulties in using remote control software are firewalls and blocked ports, as well as NAT routing for local IP addresses.
- If you use TeamViewer you don't have to worry about firewalls: TeamViewer will find a route to your partner.

Fig. 3. Subset of TeamViewer's features listed on Softpedia.

Table 1
Overview of the approach's components.

Component	Usage	Part of
Softpedia	Source of requirements used as input for our framework	External
Reqs Scraper	Automate collecting requirements from Softpedia	Framework
Reqs Filter	Filter out "non-qualified" requirements	Framework
Reqs Cluster	Group similar requirements into clusters for further processing	Framework
Attribute Extracter	Extract requirement's attributes to construct Rupp's boilerplate	Framework
Reqs Generator	To generate new requirements following Rupp's boilerplate	Framework
Candidate Selector	Apply heuristic strategies to select candidate creative requirements	Framework
Reqs Analyst	Stakeholders who perform elicitation and elaboration of software requirements	External

a 29-words description advertising an Antivirus software says, "Integrating award winning antivirus engines from 360 Cloud Scan Engine, 360 QVMII AI Engine, Avira and Bitdefender to provide you with the ultimate in Virus detection and protection capabilities". Another description from Web Browser containing 24 words "JavaScript Debugger ("Venkman") lets you debug JavaScript code on your websites, add-ons and even in SeaMonkey itself with a suite of powerful debugging tools" include around 10 technical terms. Therefore, we make a pragmatic choice and discard feature descriptions with less than 5 and more than 15 words, which results in a final filtered set of 2127 requirements out of initial 2847 requirements collected (74.7%).

2. Next, we discard non-English words and non-alphabetic letters. During the aforementioned analysis, we identify that the non-English words are typically technical keywords or typos. To our opinion, these terms are unlikely to produce meaningful attributes for a boilerplate in our context.

Table 2
A demographic overview of the requirements we have collected for the study.

Domain	# of systems	# of features		vocabulary (# of words)
		Initial	Filtered	
Antivirus	221	1087	901	3214
Web Browser	168	1286	884	2853
File Sharing	182	474	342	1714
Total	571	2847	2127	7781

3. Finally, we extract the original form or root of each word applying lemmatization (e.g., sending, sent $\Rightarrow$ send). To that end, we reduce inflectional forms and additional computational overhead to train our language model presented in Section 4.1.3.

A demographic overview of the requirements (i.e., feature descriptions in this case) we have collected for our study is presented in Table 2.

4.1.3. Clustering the requirements

In this phase, we arrange the requirements into individual clusters of highly related features. Although developers tend to introduce innovative features from time to time in an effort to differentiate their software from others in the same application domain, it is inevitable that the software systems in a domain include common features that are fundamentally similar. To that end, clustering the requirements can play a critical role in attaining combinational creativity, an approach that forms new ideas through a combination of familiar ideas (Boden, 2003).

K-nearest neighbors (KNN) (Altman, 1992) and k-means (Lloyd, 1982) are two popular algorithms widely used for clustering textual data. As we want our framework to be scalable, we leverage BIRCH (Zhang et al., 1996), a more efficient algorithm for large datasets. To that end, we expect that our framework is easily expandable to handle large-scale industry-size dataset. The process we follow for clustering is twofold. At first, we train the requirement dataset by applying doc2vec (Le and Mikolov, 2014) and obtain real-valued vectors as the output. Here, a requirement is represented by a vector and similar vector values indicate analogous requirements. Next, the BIRCH algorithm will use these vectors and group them into requirement clusters. Below we provide further details on these steps.

Training the Doc2Vec model. In order to train our model, we use gensim,[6] a popular open-source framework for natural language processing. In case of each domain, first we tokenize a feature into a list of words and feed them as input to our training model. In this manner, we train each domain separately using the parameters as follows. [Please note that the values we choose here are based on some heuristic experiments.]

1. **vector_size = 200**. It indicates the "fixed-length" size of a continuous-valued vector that represents a feature. Research has suggested a vector size in between 100 and 300 (Le and Mikolov, 2014; Mikolov et al., 2013) depending on the dataset. Since we have a dataset with a relatively small vocabulary, in order to minimize computational complexity and maintain a satisfactory performance at the same time, we pick 200 as our vector size.
2. **min_count = 2**. It indicates the minimum number of times a word must appear in the corpus so that we can include it in our training process. A word with a single occurrence gets discarded since it is less likely to add any value in building semantic similarity models for the feature set.
3. **epochs = 25**. In an ML approach, "epochs" represents the number of iterations a training session goes through. In case of a very large corpus (for example, containing millions of words), as Le and Mikolov (2014) suggest, we should use 10 to 20 iterations. For a smaller dataset like ours, however, a training session with more iterations may help us obtain a better distributed representation, where a similar feature is represented by a similar vector.

The output of this training process is a $n \times 200$ matrix M, where n is the number of requirements in a domain and each row in M represents an individual requirement. In this matrix, two semantically similar requirements are represented by comparable vectors. This characteristic is leveraged in clustering requirements, which is discussed next.

Creating requirement clusters. In order to cluster the requirements, we employ the BIRCH algorithm (Zhang et al., 1996), implemented in the scikit-learn[7] ML library. In an attempt to find the optimized number of clusters, the Silhouette values ranging from -1 to $+1$ are calculated. Here, a higher Silhouette value indicates a highly cohesive cluster, whereas a lower value indicates otherwise (Rousseeuw, 1987). In case of each domain, our clustering process starts with $k = 2$ and we keep increasing k until the Silhouette values start to decrease. In this manner, we obtain 5, 4, and 5 clusters for Antivirus, Web Browser, and File Sharing with Silhouette values of 0.468, 0.424, and 0.376, respectively.

4.1.4. Requirements generation

In this phase, our aim is to create new requirements leveraging Rupp's boilerplate (cf. Section 3.1). To that end, here, we discuss our strategy to sample requirements from clusters and further explain the techniques we apply to capture boilerplate attributes in order to formulate new requirements.

Requirements sampling. In order to examine how our framework performs in generating creative requirements, we follow a stratified sampling technique (Diez et al., 2012) for each application domain, and sample data from disjoint clusters of requirements. As Diez et al. (2012) suggests, following this approach we can obtain samples that are better representations of the requirements population. We randomly select 100 requirements from the clusters in each application domain. Due to the stratified sampling strategy (Diez et al., 2012) we follow, the number of requirements contributed by each cluster is directly proportional to its size.

New requirements creation. In an attempt to capture Boilerplate attributes, we first use **spacy**[8] to conduct POS tagging and text chunking activities. We then analyze a tagged requirement, identify a verb/verb phrase as a process, find additional attributes such as an object as a noun/noun phrase and additional details as another noun phrase. In reality, a requirement may contain more than one group of attributes. Therefore, from a group of 3 requirements, we alternatively combine the attributes extracted from them to formulate new requirements. Considering that we get 3 attributes from each requirement, we can write up to 3 verbs $\times$ 2 objects $\times$ 1 additional details $=$ 6 new requirements.

4.1.5. Selection of candidate requirements

To promote creativity as well as to keep the activities of the participants in our human-subject study (cf. Section 4.2) manageable, we need to pragmatically select a smaller set of potentially creative requirements. In this work, we refer to the items in that smaller set as *candidate requirements*. As our goal is to select a short list of requirements that can potentially promote creativity, we need to focus on two essential creativity attributes, *novelty* and *appropriateness* (Sternberg and Sternberg, 1999; Mumford, 2003). At this point, in order to accomplish our goal, we first need to put these attributes into perspectives with respect to software requirements in a real-world scenario. The *appropriateness* attribute, defined as being useful and adaptive to task constraints, of software requirements is often driven by specific software' characteristics, by project team's plan, and in a broader sense, by company vision. As these constraints vary across software and their stakeholders, retrieving candidate requirements based on appropriateness may not always be feasible. On the contrary, the *novelty* attribute, characterized as a product being original and unexpected, is not restricted by external factors. Therefore, we

6 https://radimrehurek.com/gensim/.

7 https://scikit-learn.org.

8 https://spacy.io.

Table 3
The randomly picked five automated requirements from each domain. (Each one starts with: *The system shall provide user with ability to ...*).

Antivirus	AV1	... get any background illegal sniffer hacker
	AV2	... suspend password protection shield
	AV3	... build advanced heuristic analysis malicious program
	AV4	... switch Antivirus vulnerability
	AV5	... prevent emergency situation any recovery tool
Web Browser	WB1	... export microsoft internet explorer a file
	WB2	... drop search button to the top
	WB3	... keep any user intervention recent web page
	WB4	... display functionality description
	WB5	... make file easy mouse gesture
File Sharing	FS1	... customize the files
	FS2	... set icon every downloading file
	FS3	... highlight the network user
	FS4	... save traffic download speed
	FS5	... share the client firewall

find novelty to be a promising attribute for the task of selecting candidate requirements.

Following the work presented in Bhowmik et al. (2014), we need to identify least familiar requirements that are expected to be more creative. To do so, we calculate the TF–IDF Cosine similarity scores between each generated requirement and existing requirements in the same application domain. Then, we take requirements that are at the bottom 10% of the similarity chart and randomly select five final candidates for further evaluation. Table 3 represents these candidate requirements. This randomization improves the validity of our framework's evaluation, which is detailed in the next section. Furthermore, to our opinion, five requirements from each domain would be manageable for our study participants. Next, we present the initial human subject evaluation of our framework.

4.2. Initial evaluation of the framework

4.2.1. Research questions

The main objective of this human subject evaluation is to assess how our framework performs in supporting the generation of creative requirements. To that end, we want to evaluate the automated requirements for Antivirus, Web Browser, and File Sharing in terms of their creative merits. Therefore, we ask the following research question:

***RQ*$_{1.1}$ – How creative the requirements generated by our framework are?**

At this stage, it is worth mentioning that generating text in an automated manner is inherently complex and the currently available techniques along this line are still limited, despite considerable progress made by modern NLP research. Although we have a few advanced-level versions of Rupp's boilerplate (Pohl, 2010), in our framework we have utilized a preliminary version considering its simplicity. Consequently, we anticipate that the automated statements in our case may often be incomplete, grammatically incorrect, or semantically wrong. However, our assumption is that sometimes such a statement could be interesting enough to perform as a creativity trigger (Burnay et al., 2016), that is, a collection of words or terms which inspires creative thinking among its beholders. To that end, we also want to assess this triggering aspect of a generated requirement, thereby ask the research question:

***RQ*$_{1.2}$ – How helpful the generated requirements are in capturing new requirements?**

In case of startups and small-to-medium-size software firms, typical RE activities, such as analyzing and clarifying requirements, are often performed by regular developers. As the developer community in the real world is undoubtedly diverse in terms of the experience and skill sets they possess, we would like to explore to what extent the framework's outcome depends on the developer's experience level. Therefore, the research question we ask along this line is as follows.

***RQ*$_{1.3}$ – Developers with what level of experience benefit from our automated framework?**

4.2.2. Study setup

For this study, thirty developers with a diverse background and software development expertise were recruited. The participants came from both undergraduate and graduate students and staff programmers working at our institute. We made confidentiality agreement with our participants about respecting their anonymity. Following the work on creativity in RE by other researchers, e.g., Murukannaiah et al. (2016), we wrote a questionnaire that asked the participants to rate each automated requirement on three different creativity attributes: **clarity**–"unambiguous and provides an appropriate level of detail" (Murukannaiah et al., 2016), **novelty**–"original and unexpected" (Sternberg and Sternberg, 1999), and **usefulness**–"adaptive to the system and contains value or utility" (Murukannaiah et al., 2016). The participants provided ratings on a 5-point Likert scale: 1 = very low, 2 = low, 3 = medium, 4 = high, 5 = very high. Each participant rated all 15 randomly selected requirements as shown in Table 3. We also asked the participants to provide some justifications of their ratings and rewrite/elaborate the requirement, if the idea was interesting to them. We emphasized that the rewriting activity was completely voluntary and the participants were free to ignore this task if they do not find the requirements interesting.

The latter part of the questionnaire asked the participants to rate the overall helpfulness of the automated statements in capturing new requirements. Once again, this rating was captured on a 5-point Likert scale: 1 = not at all helpful, 2 = slightly helpful, 3 = somewhat helpful, 4 = moderately helpful, 5 = extremely helpful. Please note that we requested them to provide this rating for each software domain. Furthermore, the questionnaire included queries about a participant's familiarity level with the application domains and software development experience. Before the participants started working on the questionnaire, we provided them with a short tutorial explaining different concepts critical to this study, including the rating scales and the keywords: clarity, novelty, and usefulness. We adapted the definition of our three creativity attributes from Murukannaiah et al. (2016) and further refined them to enhance understandability. The refined definitions are as follows.

1. **Clarity:** A clear requirement is unambiguous and provides an appropriate level of detail.
2. **Novelty:** A novel requirement is something that a user finds original and unexpected, i.e., something that is not common place, mundane, or conventional.
3. **Usefulness:** A useful (and implementable) requirement leads to a product that provides value or utility to its users.

Our participants worked individually and approximately spent one hour and thirty minutes, on an average, to finish the assigned tasks.

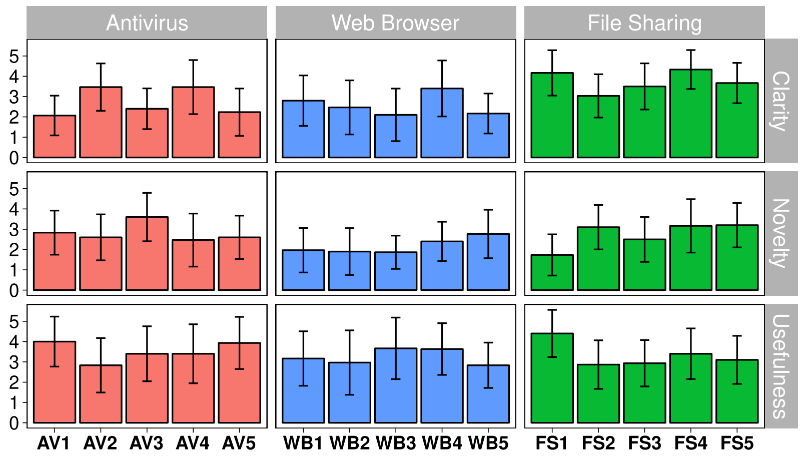


Fig. 4. The ratings for helpfulness of our generated requirements.

4.2.3. Results and analysis

$RQ_{1.1}$. ***How creative the requirements generated by our framework are?*** In Fig. 4, we show the average ratings along with associated standard errors on clarity, novelty, and usefulness for each requirement. We observe that the File Sharing requirements obtain relatively better clarity scores ranging from 3 to 4, i.e., medium to high. For the other two domains, we clearly notice some mixed opinions among the participants as just two Antivirus and one Web Browser requirements receive average ratings above three. Such ratings are probably not surprising, considering the fact that the requirements are often incomplete and grammatically incorrect. The requirements are fairly consistent across the domains, however, on their ratings for novelty and usefulness. We note that a majority of File Sharing requirements as well as a couple from Antivirus have received medium or high ratings for novelty, whereas all the requirements for Web Browser obtaining medium or even lower. We find the average usefulness ratings fairly consistent and intriguing as we see all the requirements (excluding one from Antivirus, AV2) obtaining 3 or more as average ratings (up to 4.5 for FS1). In retrospect, we do not anticipate that every requirement generated by our approach will be perfect on clarity, novelty, and usefulness. Keeping this aspect into consideration, the results discussed so far suggest that our framework often generates requirements with medium to high creativity level. Further post-hoc analysis uncovers additional insights along this line, which are discussed in Section 4.3.

$RQ_{1.2}$ – ***How helpful the generated requirements are in capturing new requirements?*** Our data exhibits 253 instances of rewritten requirements. Considering 30 participants and each with an opportunity to work on 15 requirements, we could have the maximum of 450 rewritten requirements. Therefore, 253 becomes 56.2% of the total possibilities. Given that requirement rewriting or elaboration activity was completely voluntary, we find this percentage reasonably high. It should be noted that the rewriting activity for some automated requirements may not always improve all the creativity attributes except clarity. However, considering the voluntary nature of the rewriting activity in our study, it is reasonable to consider that a beholder would spend time to rewrite/elaborate an automated requirement if it is interesting and understandable enough to draw her attention. In addition, the average ratings provided by the participants on helpfulness (cf. Section 4.2.2) of the automated statements in case of capturing new features are 3.75, 3.25, and 3.6 with the medians being 4, 3, and 4 for File Sharing, Web Browser, and Antivirus, respectively. Fig. 5 provides a box plot summarizing these ratings. Such results illustrate that the framework is capable of generating requirements that are moderately helpful in inspiring creative thinking for further elaboration. Section 4.3 includes more insights along this line, specially some interesting observations about Web Browser.

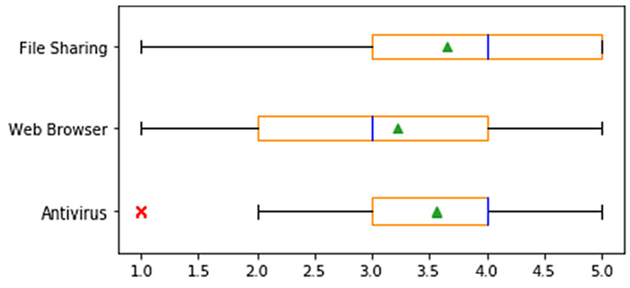


Fig. 5. Requirement usefulness ratings.

$RQ_{1.3}$ – ***Developers with what level of experience benefit from our automated framework?*** In an attempt to answer this research question, we want to examine if there exists an association between developers' level of experience and their requirement-rewriting activities (in case of each domain). Considering the professional development experience, i.e., full- or part-time software development jobs, internships, freelance development activities, and the year-long computer science or software engineering capstone projects where the participants developed software for real-world customers, we divide our participants into three different groups: **low experience** (<2 years), **average experience** (2 to 4 years), and **high experience** (>4 years). Accordingly, we obtain eight, twelve, and ten participants with low-, medium-, and high-level of experience, respectively. In case of each application domain, we find the count of rewritten requirements by the participants with each experience level. Given the optional nature of this rewriting activity, each participant had an equal likelihood of either taking or avoiding this option.

As our groups are of unequal size, in order to perform a comparison among them, we first normalize the count of rewritten requirements for each group. For example, if 8 participants contribute 26 instances of rewritten requirements for Antivirus, then what would be the count for 100 participants. In the conference version of our work (Do et al., 2019), we have performed a chi-squared test (Agresti and Kateri, 2011) along this line, which is a commonly used statistical tool for categorical data analysis. The contingency table corresponding to this test is presented as Table 4. Although, we get a χ^2 value of 28.31 (degree of freedom $df = 4$ and $p = 0.000011$), a further look into the contingency table reveals interesting implications. We observe

Table 4
Contingency table showing relationship between developer experience and requirement rewriting activity.

	Low			Medium			High			Total
	Frequency	spr	*p*-value	Frequency	spr	*p*-value	Frequency	spr	*p*-value	
Antivirus	325	1.5	.714	258	−0.9	.926	330	−0.5	.992	913
Web Browser	212	−3.4	**.021**	283	2.5	.187	320	1.0	.920	815
File Sharing	275	1.9	.438	200	−1.6	.659	270	−0.4	.996	745
Total	812			741			920			2473

that only the participants with low-level of experience have a statistically significant negative association (studentized Pearson residual $spr = -3.4$ and $p = 0.021$) with rewriting requirements for Web Browser. The rest of the cells in the contingency table, however, do not indicate any statistically significant association. In other words, the results possibly suggest that the framework would be useful for developers with different levels of experience. Section 5.1 presents a further investigation along this line.

4.3. *Interpretation of results*

4.3.1. *Mixed ratings for clarity*

Further analysis on our collected data suggests that the participants often got annoyed by issues, such as incompleteness and syntax errors, which are common to our automated statements. For example, in case of AV3: “Antivirus system shall provide the user with the ability to build advanced heuristic analysis malicious program”, a comment in the justification part reads “...looks like a good one but incomplete and awkward wording. Had to read a few times to actually get it”. This specific participant rewrites the requirement as “Antivirus system shall provide the user with the ability to generate an advanced heuristic analysis report on any malicious program”. We notice that in total 18 of the 30 participants, i.e., 60%, actually elaborated this requirement and incorporated their thoughts on malware and other malicious programs. We also observe that AV3, despite its low clarity rating, receives relative higher average scores on novelty and usefulness, 3.75 and 3.5, respectively (please see Fig. 4).

4.3.2. *Lower score for novelty*

As Maiden and colleagues suggest, the notion of creativity in RE indicates “the capture of requirements that are *new to project stakeholders* but may not be *historically new to humankind*” (Maiden et al., 2010). Accordingly, the perceived novelty aspect of a requirement often gets influenced by the system under consideration. In other words, we may find a feature to be novel for a certain software system, whereas, the same functionality might already exist in some other software. In our initial study, we ask the participants to rate requirements for application domains, not for an individual software. Furthermore, our participants are regular users of multiple software systems from all three domains. We posit that such familiarity, at least to some extent, limits the level of surprise, particularly in case of a well known domain, such as Web Browser. We find a comment from one of the participants about WB4, “...the browsers I know of display functionality description when I hover the mouse on an icon”. We observe that the participants were often cautious on novelty as well as usefulness scores, despite their excitement about the idea was clearly evident in comments. Let us take FS5 “System shall provide the user with the ability to share the client firewall” for an example. Some common comments about this requirement include “useful in the right situation..., fairly novel, but not sure about the potential security risks.... would not rate very high”. On a different note, such observations further imply our automated requirements’ ability to provoke critical thinking among beholders.

4.3.3. *Rewriting the requirements*

Each of the 15 requirements was further elaborated where some received more attempts than the others. Among the more frequently elaborated requirements, we find AV1, AV5, and WB2 who are rewritten by 24, 22, and 22 participants, respectively. An additional analysis uncovers an interesting trend. Each one of these three requirements are ranked low both on clarity ($\approx$2) and novelty (ranging from 1.9 to 2.93). WB2 “Browser shall provide the user with the ability to drop search button to the top”, is in fact among the lowest rated automated requirement in our study (average rating $\approx$2.67). However, many participants still took time to think about this idea and some of the elaborations look striking. An intriguing example would be: “The browser shall include a permanent drop-down search bar so that users can easily search the internet at any given time”. The requirement suggests that the browser could add an integrated panel accommodating a search bar so that a user may search the internet without going away from the website she is currently browsing. Arguably, if this feature is implemented, we can enjoy the advantage of having double monitors. This observation points out some interesting questions: Is the framework indeed able to promote creativity even for a matured domain? Even if the overall creativity merits of some automated requirements are low, could they still serve as good starting points for further innovative features? With an aim to gain additional insights along this line, we conduct additional analysis and study for this invited extension, which is discussed in the next section.

5. Second study: Further exploring the framework’s benefits

Building upon the findings of our initial study presented in the previous section, in this extended work, we further explore our framework’s potential benefits by addressing the following important questions.

- Will our framework be beneficial to developers with various levels of experience?
- How the framework could support creativity for a matured application domain?
- Even when our framework produces requirements with low creativity merits, would they still be useful to stakeholders?

In order to answer these questions and to uncover valuable implications, we conduct relevant qualitative and quantitative analyses and carry out an additional study that are detailed in the next section.

5.1. *On developers with different experience levels*

Motivated by the aforementioned indication, in this invited extension, we plan to conduct further investigation and strengthen the answer for $RQ_{1.3}$. The rest of this section detail this investigation along with the results and analysis.

Generalized Linear Models
Outcome = Intercept + (Fixed Effects) + Error

Mixed Models
Outcome = Intercept + (Fixed Effects) + (Random Effects) + Error

Fig. 6. General versus Mixed-Effect Model.

5.1.1. *Hypothesis*

To reiterate, we are interested to know if the requirements generated by our approach are beneficial (i.e., helpful or useful) to only specific groups of developers with certain level of experience or they can act as creativity triggers for a wide range of developers (irrespective of their experience). In other words, if our approach is generally beneficial, whether a developer finds it useful or not should not depend on her development experience. With an objective to explore this notion, we formulate the following *null* and *alternative* hypotheses.

H_0: Developer's software development experience has no influence on whether they find the generated requirements useful.

H_1: Developer's software development experience influences whether they find the generated requirements useful.

During the earlier human-subject study, we clearly explained the participants that the activity of rewriting candidate requirements were completely voluntary and they could ignore this task if they did not find an idea interesting and worth elaborating. Therefore, in this context, we presume that the generated requirements are useful or helpful if they prompt our participants to rewrite/elaborate the given items. Accordingly, we operationalize usefulness as the act of rewriting a given requirement, i.e., if rewritten then useful, else not useful. As software development experience, we consider full- or part-time software development jobs, internships, freelance development activities, and the year-long computer science or software engineering capstone projects where the participants developed software for real-world customers. With such constructs, we conduct a mixed-effect logistic regression analysis with repeated-measures design in order to test our hypothesis. The next few paragraphs provide some background on repeated-measures design and mixed effects logistic regression along with the rationale behind choosing this analysis technique.

5.1.2. *Repeated-measures design*

It is an experimental design strategy when we take multiple measures of the same subject under different treatment conditions or over multiple time periods (Bagiella et al., 2000). Repeated-measures design is particularly useful when the study has a limited number of participants as it reduces the variance of estimates among subjects. Therefore, this design allows researchers to draw statistical inferences with a relatively small set of subjects (Bagiella et al., 2000).

5.1.3. *Mixed effects logistic regression*

Linear mixed effects model is an extension of generalized linear models used to measure effects in regression models (Bates and Pinheiro, 1998). Unlike linear models whose source of variance comes from the random samples we collect the data from, a mixed effects model's variance is contributed by the variables themselves (Winter, 2013). As depicted in Fig. 6, a generalized linear model contains intercept, *fixed effects*, and an unknown error term, whereas the mixed-effect model includes additional *random effects* stemmed from the variability within the variables.

Despite the widespread use of mixed effects models in various disciplines, there have been distinctly conflicting definitions of *fixed* and *random effects* (Kreft and De Leeuw, 1998; Snijders, 2011; Searle et al., 2009). In this paper, we use those defined in Winter (2013), which are not only simple and straightforward but also preserving important attributes established by the literature. According to Winter (2013), fixed and random effects can be distinguished by the nature of data levels (Winter, 2013). In this study, we treat a variable as a fixed effect factor if the collected data contains all levels of interest for that variable (e.g., genders: male and female; ratings: low, medium, high, etc.). On the other hand, with random effects, we may not collect all possible levels. Instead, we sample from "the population of interest" (Winter, 2013). For instance, the subjects in our sample (i.e., the participants we have recruited for our study) are far from representing the entire population of software developers. However, the goal is to generalize our statistical inferences for a broader range of population. Therefore, as an alternative, we can also think of fixed and random effects in terms of how they influence the data. Fixed effects are expected to have a structured and predictable influence, whereas random effects contribute to non-systematic and unpredictable influences on the data (Winter, 2013).

In sum, inclusion of random effects in mixed effects models allows us to draw conclusions with increased confidence even if the data is collected from a relatively small sample. This is one of the reasons we consider a mixed effects model for our analysis. At this point, it should be noted that the difference between fixed and random effects are loosely defined (Gelman et al., 2005). To that end, determining which effect a variable belongs to depends on what our research question is and what statistical inference we are planning to draw to answer the research question (Gelman et al., 2005).

Another important reason for us to consider mixed effects model is that in order to properly employ linear models, several assumptions must be met to guarantee accuracy and consistency of the analysis. A principal one is the *independence assumption* stating that the collected data/observations should be independent, i.e., the value of one observation should not be affected by other observation (e.g., data points comes from different subjects). Since our data collection involves repeated-measures with multiple responses gathered from the same subject, a mixed-effect model is a reasonable choice to resolve violation of independence assumption, thereby making our statistical inferences robust and reliable.

5.1.4. *Model setup*

In order to test our hypothesis on the relation between the developers' software development experience and their act of rewriting the requirements, the latter is the dependent variable with two possible values: rewrite the requirement or do not rewrite the requirement. Therefore, we develop a mixed effects logistic regression model, a specific type of linear regression model where the dependent variable is dichotomous (i.e., it can take only two possible values, e.g., gender: male/female, tumor diagnosis: yes/no, rewritten requirement: yes/no). We define the variables of our logistic regression model in Table 5.

In our study, all the participants were asked to voluntarily rewrite 5 requirements from 3 software domains adding up to 15 requirements per participant. Certainly, our model needs to factor in the variability contributed by the inherent variability among the subjects/participants. For instance, every participant may have slightly different motivation that is likely to affect his/her decision when it comes to rewriting those requirements (e.g., requirements come from familiar domain, participant is likely to rewrite first few requirements and become lazy with the remaining ones, etc.). This motivation is non-systematic and unpredictable, therefore constituting a ***random effect*** for subjects and "characterizing idiosyncratic variation due to individual differences" (Winter, 2013). Furthermore, when a participant's attention move from requirement to requirement, the likelihood of

Table 5
Our mixed effects logistic regression model's variables.

Variable	Description	Effect	Variable type
subject_id	Participant in the study	Random	Categorical - Independent
requirement_id	Requirement in the study	Random	Categorical - Independent
experience	Months of working experiences in software development	Fixed	Continuous - Independent
domain	Software domains (Antivirus, Web Browser, File Sharing)	Fixed	Categorical - Independent
rewritten	Binary variable stating whether participants rewrote requirements or not	N/A	Categorical - Dependent

```
model <- glmer(data=data, rewritten ~domain + experiences +
          (1|subject_id) + (1|requirement_id), family = binomial,
          control = glmerControl(optimizer = "bobyqa"))
```

Fig. 7. R command used to run our linear mixed effects model.

rewriting a requirement can be randomly affected by the variability among the requirements. Therefore, we consider requirement as another random effect in our model. As the ***fixed effects*** in our model, we enter software development experience in months and software domain (without interaction). Although in our study we consider only three software domains, obviously there are more domains in the real world. However, we *operationally define* domain in the context of our study as the difference among Antivirus, Web Browser, and File Sharing, and we "exhaustively" test all of them, i.e., each participant gets the opportunity to work on all the domains in an equal manner. In other words, we fully exhaust the factor "domain" (as we defined) and following Winter (2013), we do not treat domain as a random effect. Each subject and requirement are assigned individual ID so that the mixed model can recognize them as random effects and treat them differently than the fixed effects. Outcome variable is encoded as a binary variable taking Yes/No values, where "Yes" means a requirement has been rewritten by the participant in the same observation and "No", otherwise.

We use *R* (R Core Team, 2018) and *lme4* package (Bates et al., 2015) to perform the linear mixed effects analysis. In particular, we run our model using **glmer** function from **lme4** package (Bates et al., 2015) with **domain** as requirement level categorical predictor (Antivirus, Web Browser, or File Sharing), **experience** as subject level continuous predictor, and two random effect by **subject_id** and **requirement_id** (cf. Fig. 7).

5.1.5. Results and analysis

Table 6 presents the results of our mixed effect logistic regression model for rewritten requirements. Here, we see the associated values for estimate, some of square (SE) and *p*-value for all the parameters in the model. It should be noted that we build the logistic regression model considering Antivirus as the base which is indicated by the parameter "intercept" in Table 6. In order to test our hypothesis, we are particularly interested in the experience parameter of the model. As Table 6 suggests, the estimate and SE values for experience are −0.007 and 0.034, respectively, with a *p*-value of 0.827, which is much higher than $\alpha = 0.05$. These results indicate that *we do not find any evidence* at $\alpha = 0.05$ that experience acts as a factor for a developer's requirement rewriting activity. Therefore, we fail to reject our null hypothesis. In other words, based on the statistical evidence, we accept:

> Developer's software development experience has **no influence** on whether they find the generated requirements useful.

This finding further supports the initial suggestion, provided by the χ^2 test we presented earlier, that our framework can be beneficial for a wide range of developers with various experience levels.

Table 6
Mixed effects logistic regression for rewritten requirements.

Parameter	Est.	SE	*p*-value
Intercept	0.791	0.650	0.224
Domain File Sharing	−0.933	0.463	0.043*
Domain Web Browser	−0.415	0.461	0.368
Experiences	−0.007	0.034	**0.827**
Subject	–	2.359	–
Requirement	–	0.54	–
$N_{subjects} = 30$		$N_{requirements} = 15$	

5.2. On software domain and creative merits

Motivated by the insights from our initial human subject evaluation presented in the earlier sections, here, we focus on further examining our framework on two specific aspects highlighted in Section 4.3. There, we have indicated that the level of surprise can play an integral part in the perceived creative merit of a requirement. As Filipowicz (2006) points out, the level of surprise positively affects the performance of a creative activity. In real world, some software systems, or application domains for that matter, have been around for quite some time. Furthermore, some application domains (e.g., Web Browser) are widely used by a large and diverse group of users, receive frequent contribution by many open source software developers worldwide (e.g., Mozilla Firefox), and include multiple highly competitive software systems striving to attract more users. Such aspects can make an application domain highly saturated in that the software systems in the domain might already have a wide range of features that are well known to common people. Therefore, the requirements (or triggers) automatically generated by our framework may have an inherent limitation on their level of surprise. It would be interesting to examine if our framework can be useful for a saturated application domain. To that end, the first research question we ask in this study:

> ***$RQ_{2.1}$* – Can our framework promote creativity for a matured application domain such as Web Browser?**

In our initial study, we noticed that some automated requirements consistently received low ratings, especially on novelty (e.g., AV1, WB1, and FS1), from the participants. However, given our study setup, each requirement had an equal probability of being elaborated/rewritten by the participants. Therefore, as a natural followup on the aspect "level of surprise", the second research question we formulate:

> ***$RQ_{2.2}$* – Can the automated requirements with low creative merits further promote innovative features?**

5.2.1. Study setup

In order to answer the aforementioned research questions, we follow a setup similar to our initial study (cf. Section 4.2.2) in that we collect ratings for clarity, novelty, and usefulness at a 5-point Likert scale. However, in this invited extension, we focus on the rewritten requirements (elaborated from the automated requirements/triggers by the participants in our initial study). We recruit

Table 7
Ratings: Automated vs rewritten requirements.

		Automated Reqs. (by students & profs.)		Rewritten Reqs. (by domain expert)	
		Mean	SD	Mean	SD
Antivirus	Clarity	2.66	1.22	4.42	0.75
	Novelty	2.90	1.27	3.01	1.01
	Usefulness	4.07	1.11	4.69	0.72
Web Browser	Clarity	2.98	1.21	4.79	0.61
	Novelty	2.15	1.07	3.42	0.92
	Usefulness	3.92	1.04	3.88	1.01
File Sharing	Clarity	3.61	1.21	3.89	0.83
	Novelty	2.62	1.28	3.01	0.83
	Usefulness	3.85	1.04	3.21	0.84

three participants Bob, Mary, and John (pseudonim used) who are domain experts in security, Web Browser, and File Sharing, respectively. We call them domain experts for several reasons. First, the participants have 5–7 years of full-time professional software development experience. Second, Bob has worked as a cyber security analyst for 3 years at a security focused software firm and possesses multiple security certificates, whereas Mary and John use multiple Web Browsers and File Sharing software, respectively, on a daily basis and an active contributor to at least one open source software in the domain. Last but not the least, each participant has a Master's degree in Computer Science. At the beginning of the session, we provide the participants with a short tutorial explaining different concepts critical to this study, including the rating scales and the keywords: clarity, novelty, and usefulness. Each participant works individually for about one and a half hours and provides ratings for the rewritten requirements from the domain he/she is the expert in. Similar to the initial study, the participants also provide justifications of their ratings, if needed.

In this study, one of our objectives is to gain an increased confidence in the findings about the utility of our framework. To that end, we want to obtain a conservative estimate on the creative merits of the rewritten requirements. Since, domain experts generally have a deeper knowledge about the software systems serving the domain and the functionalities they provide, the level of surprise for and expert about a requirement would be limited. Therefore, following Maiden et al. (2004a,b), Maiden and Robertson (2005) and Karlsen et al. (2009), in this study, we collect expert-ratings, which will give us that lower estimate that can potentially reflect the minimum level of utility our framework can provide.

5.2.2. Results and analysis

$RQ_{2.1}$ ***– Can our framework promote creativity for a matured application domain such as Web Browser?*** In our initial study, out of 253 instances of rewritten requirements, we notice 88, 85, and 80 instances for Antivirus, Web Browser, and File Sharing, respectively. The corresponding domain expert evaluates these requirements and provides ratings. Considering the voluntary nature of our rewriting activity, those numbers clearly indicate that the triggers for the Web Browser domain, despite possible limitations on the level of surprise, have inspired the act of rewriting at a level comparable to the other domains.

Table 7 presents the means (i.e., average in this case) and standard deviations (*SD*s) for clarity, novelty, and usefulness ratings obtained from the experts for the rewritten requirements. This table also includes ratings (means and *SD*s) for the triggers (i.e., the automated requirements), belonging to the corresponding rewritten ones, obtained in our initial study. For Web Browser, we notice that both the means of clarity and novelty (4.79 and 3.42) for rewritten requirements are much higher than those of Antivirus and File Sharing. In fact, the mean novelty of 3.42 is the highest among the three domains. In terms of usefulness, Web Browser (mean 3.88) does much better than File Sharing (3.21) but not as good as Antivirus (4.69). Considering the comparison presented in Fig. 8, we can see that the overall ratings for the rewritten requirements in case of Web Browser increase in a similar manner, specially for clarity and novelty, as for the other domains. This analysis suggests that the automated requirements for Web Browser generated by our framework **inspires creative muse** among the participants in a manner similar to the other domains. Therefore, we conclude that our framework promote creativity even for a matured application domain such as Web Browser.

Here, we would like to point out that there is an exception in the usefulness score which drops by .04 (a 1% drop, which is negligible in our opinion) for Web Browser. We also observe a 16.6% drop in usefulness for File Sharing. To our opinion, this drop in the average usefulness ratings by experts is not that surprising. As the domain experts generally have a better idea about the implementation constrains, they can foresee certain technical difficulties that may make a requirement infeasible for implementation (at least in the current context). A comment from Mary, our Web Browser expert, corroborates this aspect, "This feature sounds like a great idea... I will rate 5 for novelty. But I know that we cannot implement it in its current form. 3 for usefulness".

$RQ_{2.2}$ – **Can the automated requirements with low creative merits further promote innovative features?** In order to answer this research question, we want to concentrate on the automated requirements/triggers those received particularly low ratings and further investigate the expert ratings for their rewritten counterparts. In this context, we operationalize any rating less than 3 as a low rating. Our objective is to see if the creativity ratings for the rewritten requirements has significantly improved compared to the original triggers. Here, the rationale is that such an increase would indicate the triggers' ability to provoke brainstorming activities in the developer's mind, thereby promoting creative requirements. To that end, we formulate the following *null* and *alternative* hypothesis.

H_0: The overall creativity ratings for the automated requirements are not less than the ratings for their rewritten counterparts.

H_1: The overall creativity ratings for the automated requirements are less than the ratings for their rewritten counterparts.

Before we move on to testing the above hypothesis, we would like to pay attention to an important nature of our data. It should be noted that we are going to make a comparison between the ratings for automated and rewritten requirements. The automated ones are low on clarity due to the inherent limitations of automated text generation techniques and due to the simplicity of the boilerplate we used. Whereas, the written ones are written by humans, therefore, naturally receive much higher clarity ratings compared to their automated counterparts in a consistent manner. Table 7 and Fig. 8 both can speak for this phenomenon. In this scenario, including clarity ratings to test our hypothesis may significantly bias the results towards the rewritten requirements. Therefore, in this analysis, we include only the novelty and usefulness ratings.

Considering the overall creativity ratings (novelty and usefulness combined), we find that the automated requirements are rated much lower than the corresponding rewritten requirements (mean 2.33 and 3.54 respectively). Observing the non-normal nature of our data, we conduct a Mann–Whitney-Wilcoxon test (Mann and Whitney, 1947; Wilcoxon, 1945), which is a non-parametric equivalent of the t-test (Agresti and Kateri, 2011), to statistically examine the difference between the creativity

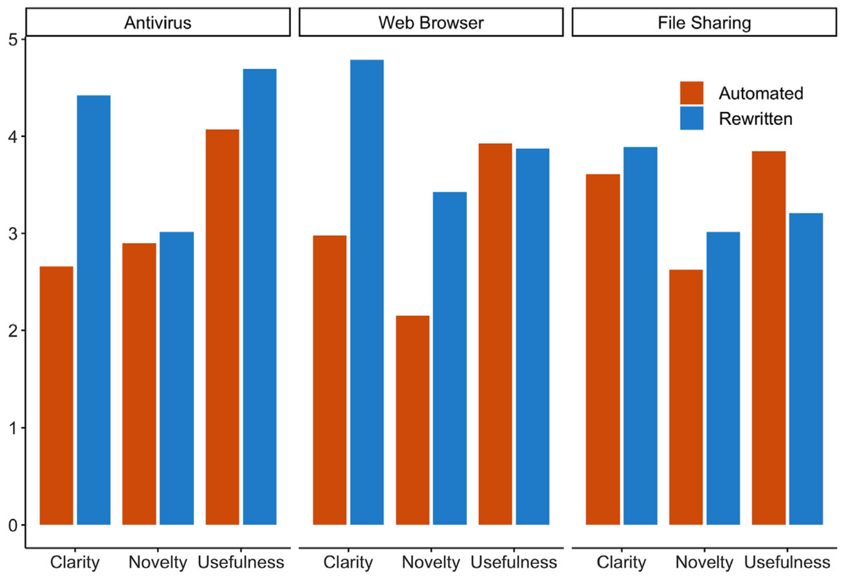


Fig. 8. Comparison of the average ratings for automated and rewritten requirements.

ratings (novelty and usefulness only). We use R (Tricentis, 2019), a popular software package for statistical computing, to perform this test. We find a W value of 144.5 with p-value < 0.00001 and effect size $r = 0.54$ (showing medium strength Cohen, 2013), which clearly indicates a statistically significant difference between the automated and rewritten requirements at $\alpha = 0.05$. In order to obtain further confidence in our finding, we conduct Mann–Whitney-Wilcoxon test (Mann and Whitney, 1947; Wilcoxon, 1945) on novelty (mean: 1.54 for automated and 3.09 for rewritten) and usefulness (mean: 1.68 for automated and 3.5 for rewritten) separately and get similar results (p-value 0.0018 and < 0.00001 for novelty and usefulness, respectively). This analysis suggests us to reject the null hypothesis, i.e., the overall creativity ratings for the rewritten requirements are indeed higher than the ratings for automated requirements they originated from. Therefore, we conclude that *the automated requirements, generated by our framework, that have low creative merits can further promote innovative features.*

Reflecting on our observation about rewriting requirements discussed in Section 4.3.3, the average novelty scores for rewritten requirements originated from AV1, AV5, and WB2 substantially increases (ranging from 3.52 to 3.93) compared to their automated counterparts (ranging from 1.9 to 2.93). The novelty scores also exhibit a similar trend (ranging from 4.33 to 4.77 for the rewritten requirements compared to the range 2.97 to 4.00 for the automated ones). These findings further attest that more innovative features can be inspired even by the automated requirements that have low ratings for novelty.

6. Implications for practitioners

6.1. On software developers with various levels of experience

As shown by the mixed-effect logistic regression model (cf. Section 5.1), the requirements generated by our approach are beneficial to a wide range of developers regardless of their work experience. The participants have few months to up to 8 years of software development experience. Nevertheless, most of them engaged in requirement-rewriting activity despite this activity being completely voluntary. This encouraging finding suggests that software developers with diverse backgrounds can, to a certain degree, take advantage of our framework. For instance, inexperienced developers can utilize our framework when they know the application domain for which they want to develop a software, however, are not really sure about the features they want to include in the application. Generated requirements from our framework can give them ideas to choose from and elaborate on in order to capture interesting features. Experienced developers or teams, on the other hand, are often under pressure from the new releases by the competitors who equip their applications with many impressive features. As demonstrated in Section 5.2, $RQ_{2.2}$, such developers or teams can take generated requirements from our framework as inspiration to capture novel ideas for their application features.

6.2. On matured domains and requirements with low creativity merits

Matured software domains, such as Web Browser or Antivirus, are expected to engage in intense competition where software firms are contentiously trying to attract users' interest. Therefore, it is crucial for a software firm to have ideas as early as possible so that it can be ahead of its competitors when it comes to analyzing, evaluating, prototyping,a and eventually delivering new features. The results discussed in Section 5.2.2, also suggest that even for requirements or triggers with low creativity merits, developers, with a bit of brainstorming, can turn seem-to-be-impractical requirements to usable instances. Which further emphasizes that our framework could clearly be helpful to developers building software for a mature application domain.

6.3. On framework usage

In this work, we have applied our framework on three popular software domains: Antivirus, Web Browser, and File Sharing. Although the current implementation of our framework supports candidate creative requirements generation for these three domains, it could be adopted for other domains available on Softpedia after necessary modifications. At present, we can run the framework by executing a python script supplying arguments, such as chosen domain (i.e., Antivirus, Web Browser, or File Sharing) and the number of requirements to generate. In future, we plan to implement an user-friendly web application for the whole process of our framework as depicted in Fig. 2. Nevertheless, the approach presented in this paper can be beneficial for stakeholders such as requirements analysts, software developers and customers in various ways which are highlighted next.

Requirements Analysts. Our framework can be helpful to a requirements analyst in determining the future functionalities of the project that could be of rising demands based on the generated ideas. Being exposed to candidate creative requirements, a requirements analyst will be able to evaluate and analyze numerous feature ideas, thereby getting an opportunity to capture interesting features for the next software releases. By planning ahead of time, the requirements analysis and project team will be able to keep a balance on both implementing new features and meeting business requirements.

Software Developers. As discussed earlier in the paper, software developers with diverse backgrounds can greatly benefit from our framework when they seek for new feature ideas. In addition, by gaining access to ideas at an early stage, developers can swiftly implement a prototype to demonstrate to other stakeholders, such as the strategic decision makers in the firm, users, and potential customers, thereby shortening the total development time in general.

Users and Customers. Our framework can be used to suggest new features to users and potential customers. If they are interested in any of those features, they can discuss with requirements analysts and software developers on the feasibility of including the functionalities in the software (after some modifications, if needed). To that end, the users and customers will have more feature options to consider rather than relying on the development team to propose new features. In addition, with the negotiated features at hand, relevant stakeholders can work on estimating the cost associated with the implementation and the customers can receive a more accurate projection on the release timeline.

7. Threats to validity

In this paper, we have presented an automated framework to promote creativity in RE leveraging ML techniques, and requirements reuse. Furthermore, we have conducted two human subject studies evaluating our framework's ability to support creative requirements generation. However, our work has its limitations and several factors can affect the validity of this research: construct validity, internal validity, external validity, and reliability (Yin, 2013). In what follows, we elaborate on these threats to validity of our overall work.

7.1. Construct validity

Construct validity concerns to what extent the correct operational measures for the concepts being studied (Yin, 2013) are established. A major construct in this work, more specifically in our human subject studies, include measuring the theoretical construct of creative merits of both the automated and rewritten requirements. In order to capture this construct, we follow a well-established Likert scale rating for granular level creativity attributes: clarity, novelty, and usefulness for both the studies (inspired by the work of Murukannaiah et al., 2016). Furthermore, we notice corroborating evidence for our findings uncovered through further qualitative analyses (cf. Sections 4.3 and 5.2.2). We also provide tutorials with practice exercises to our participants at the beginning of the study sessions in order to train them with the notion of clarity, novelty, and usefulness from a creativity perspective. Therefore, we believe that our work holds construct validity along this line.

The nature of $RQ_{2.2}$ (cf. the second study), however, requires us to compare automatically generated text to manually written statements. As the automated text is bound to be consistently low on clarity due to inherent limitations, we believe this attribute does not capture the right construct in the given context. Rather, it may have significant bias towards the manually written statements and further confound our analysis. Therefore, we exclude clarity while addressing $RQ_{2.2}$, which, to our opinion, further improves the construct validity aspect of our work.

Developer experience is another important construct in our work. We consider true experience, captured in years and months, gained only through real world software development. In case of the second study, to find a domain expert, we take into account professional software development experience and other relevant activities, such as working as a security analyst and contributing to open source projects, that can play a critical role in improving domain knowledge. Therefore, we believe our work does not possess a significant threat to construct validity in this area.

7.2. Internal validity

Internal validity establishes the accuracy of conclusions drawn upon cause and effect (Cozby and Bates, 2012). We examine our research questions and draw conclusions based on both quantitative and qualitative analysis. We employ state-of-the-art sophisticated statistical analysis techniques, when appropriate. For instance, in order to gain additional confidence in our initial answer to $RQ_{1.3}$ obtained from the χ^2 test, we conduct a mixed effects logistic regression analysis (Bates and Pinheiro, 1998; Winter, 2013) that takes into account random variabilities originated from human participants and automated requirements. In case of $RQ_{2.2}$, following the non-normal distribution of our relevant data, we conduct Mann–Whitney-Wilcoxon test (Mann and Whitney, 1947; Wilcoxon, 1945), a non-parametric equivalent of the t-test (Agresti and Kateri, 2011). Although we did not use clarity for $RQ_{2.2}$ in order to improve construct validity, we consider such ratings to answer $RQ_{2.1}$. The rationale being in the later case is that we are interested in the trend of change between the ratings for the automated and rewritten requirements, not in the absolute difference in their cumulative means. Considering these aspects, we believe our conclusions possess reasonable internal validity.

7.3. External validity

External validity concerns establishing the domain to which a study's finding can be generalized (Yin, 2013). A major limitation of our framework is that it heavily depends on the availability of a large number of feature descriptions that are clearly and correctly written. As we leverage Rupp's boilerplate (Pohl, 2010) that requires further contexts for objects to automatically formulate requirement statements (cf. Section 3.1), our framework will not be able to perform with inputs that are brief product highlights written using a few key words and discrete terms or phrases (which is a practice commonly followed on Google Play). For demonstration, our framework is applied on three different application domains, including Antivirus, web browser, and File Sharing. In addition, we randomly pick automated requirements from each domain to evaluate performance. Therefore, if the feature descriptions for an application domain or a software system hold the conditions we just mentioned, our framework is expected to exhibit similar or better performance.

7.4. Reliability

Reliability of a study suggests that the operations of the study can be reproduced with the same results (Yin, 2013). With an aim to tackle the built-in complexity we face in automated text generation, our framework utilizes a simplified boilerplate (Pohl, 2010) that accommodates a maximum of six placeholders. An imminent consequence of this choice is that our framework often leads to incomplete and unstructured sentences (for example, AV5 and WB3). Due to this reason, the clarity aspect of the requirements are sometimes affected. We acknowledge that this is another major limitation of the framework. However, we expect that this limitation would be minimized if we implement an enriched boilerplate. Another important point is that the algorithms we have implemented treat frequently observed contextual text as additional details for objects (please refer to Section 3.1). To that end, the element of surprise gets compromised to some degree. Nevertheless, an implementation of our framework realizing the same algorithms using the same parameters and a demonstration on the same set of original feature descriptions, to our opinion, would provide similar results.

As we pointed out in Section 5.2.2, some automated requirements from our framework might look interesting and novel, however, their implementation might be infeasible or likely to be infeasible in the current context. Such instances might warrant additional effort from the analyst to make a clear judgment. We admit this to be a limitation of our framework in that some requirements might require additional feasibility analysis. In such a case, we need to rely upon the analyst's judgment as he/she might decide to move on to the next requirement ignoring the apparently infeasible one.

To construct a manageable candidate requirements set for our human-subject study, we pragmatically assume that less familiar requirements lead to more creative requirements. Even though we have not conducted a dedicated study to validate this assumption, our previous work on RE creativity (Bhowmik et al., 2014, 2015) have suggested that requirements with lower TF–IDF scores tend to obtain higher ratings on creativity aspects. Note that our primary goal is not to claim that less familiar requirements imply enhanced creativity. Instead, we utilize this practice as a sensible way to reduce overhead for participants of the study with the main objective to show the effectiveness of our framework on generating creative requirements. Therefore, we anticipate similar results will be obtained using the assumption.

In terms of the human subject evaluations, the results from our second study are limited in that we have obtained ratings from only one expert for each application domain. However, based on some justifications the experts provided on their ratings, we do not expect a surprising difference in the overall results if the second study is conducted with more expert participants.

8. Conclusion

In this paper, we present a novel framework that provide an automated support for innovating requirements by reusing existing requirements from similar software systems and leveraging ML techniques. We also report an initial human subject evaluation of our framework using feature descriptions from three application domains and the results demonstrate the framework's ability to generate creative requirements. Moreover, additional qualitative analysis uncovers some intriguing questions about our framework for further investigation that motivate us to conduct a follow-up study.

In sum, the results we have obtained from these studies substantiate that the framework promotes creative thinking among developers with various experience levels, it has the ability to promote creativity even for a relatively matured software domain, and it may boost the innovative aspects through refined requirements originated from automated ones.

9. Future work

In future, we plan to further refine our automated framework by supporting more advanced boilerplates (e.g., EARS Boilerplate Mavin et al., 2009) that are expected to offer more complete and ready-to-use requirement instances. Furthermore, we intend to conduct an in-depth human-subject study to assess the effectiveness of our automated framework against traditional creativity techniques in RE, such as group brainstorming and use of creativity triggers in terms of individual words or short phrases representing specific qualities of an innovative product, including Convenience or Durable (Burnay et al., 2016). Such a study will help us to strengthen our observation made in this study and collect valuable feedback and insights from a diverse group of participants.

CRediT authorship contribution statement

Quoc Anh Do: Conceptualization, Methodology, Writing - original draft, Software. **Tanmay Bhowmik:** Conceptualization, Methodology, Supervision, Writing - review & editing. **Gary L. Bradshaw:** Formal analysis, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Agresti, A., Kateri, M., 2011. Categorical Data Analysis. Springer.
Altman, N.S., 1992. An introduction to kernel and nearest-neighbor nonparametric regression. Amer. Statist. 46 (3), 175–185.
Arora, C., Sabetzadeh, M., Briand, L., Zimmer, F., 2016. Automated extraction and clustering of requirements glossary terms. IEEE Trans. Softw. Eng. 43 (10), 918–945.
Bagiella, E., Sloan, R.P., Heitjan, D.F., 2000. Mixed-effects models in psychophysiology. Psychophysiology 37 (1), 13–20.
Barron, F., 1955. The disposition toward originality. J. Abnorm. Soc. Psychol. 51 (3), 478.
Bates, D., Mächler, M., Bolker, B., Walker, S., 2015. Fitting linear mixed-effects models using lme4. J. Stat. Softw. 67 (1), 1–48. http://dx.doi.org/10.18637/jss.v067.i01.
Bates, D.M., Pinheiro, J.C., 1998. Linear and nonlinear mixed-effects models. In: Conference on Applied Statistics in Agriculture.
Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C., 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3 (Feb), 1137–1155.
Berwick, R.C., Abney, S.P., Tenny, C., 1991. Principle-Based Parsing: Computation and Psycholinguistics, Vol. 44. Springer Science & Business Media.
Bhowmik, T., Niu, N., Mahmoud, A., Savolainen, J., 2014. Automated support for combinational creativity in requirements engineering. In: Proceedings of the International Requirements Engineering Conference, RE. pp. 243–252.
Bhowmik, T., Niu, N., Savolainen, J., Mahmoud, A., 2015. Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering. Requir. Eng. 20 (3), 253–280.
Boden, M.A., 2003. The Creative Mind: Myths and Mechanisms. Routledge.
Brill, E., 1992. A simple rule-based part of speech tagger. In: Proceedings of the Workshop on Speech and Natural Language. pp. 112–116.
Burnay, C., Horkoff, J., Maiden, N., 2016. Stimulating Stakeholders' imagination: New creativity triggers for eliciting novel requirements. In: Requirements Engineering Conference (RE), 2016 IEEE 24th International. IEEE, pp. 36–45.
Cohen, J., 2013. Statistical Power Analysis for the Behavioral Sciences. Routledge.
Corazza, G.E., 2016. Potential originality and effectiveness: The dynamic definition of creativity. Creat. Res. J. 28 (3), 258–267.
Cozby, P.C., Bates, S.C., 2012. Methods in Behavioral Research.
Cropley, D., Cropley, A., 2005. Engineering creativity: A systems concept of functional creativity. In: Creativity Across Domains. Psychology Press, pp. 187–204.
Diez, D.M., Barr, C.D., Cetinkaya-Rundel, M., 2012. OpenIntro Statistics, Vol. 12. CreateSpace.
Do, Q.A., Bhowmik, T., 2018. Automated generation of creative software requirements: a data-driven approach. In: Proceedings of the 1st ACM SIGSOFT International Workshop on Automated Specification Inference. ACM, pp. 9–12.

Do, Q.A., Chekuri, S.R., Bhowmik, T., 2019. Automated support to capture creative requirements via requirements reuse. In: International Conference on Software and Systems Reuse. Springer, pp. 47–63.
El-Sharkawy, S., Schmid, K., 2011. A heuristic approach for supporting product innovation in requirements engineering: a controlled experiment. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 78–93.
Farfeleder, S., Moser, T., Krall, A., Stålhane, T., Omoronyia, I., Zojer, H., 2011. Ontology-driven guidance for requirements elicitation. In: Extended Semantic Web Conference. Springer, pp. 212–226.
Filipowicz, A., 2006. From positive affect to creativity: The surprising role of surprise. Creat. Res. J. 18 (2), 141–152.
Ford, D.Y., Harris, J.J., 1992. The elusive definition of creativity. J. Creat. Behav.
Gelman, A., et al., 2005. Analysis of variance—why it is more important than ever. Ann. Stat. 33 (1), 1–53.
Google, 2020. Google chrome. https://www.softpedia.com/get/Internet/Browsers/Google-Chrome.shtml, (Accessed 28 March 2020).
Groen, E.C., Doerr, J., Adam, S., 2015. Towards crowd-based requirements engineering a research preview. In: Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 247–253.
Horkoff, J., Maiden, N.A., 2015. Creativity and conceptual modeling for requirements engineering. In: Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality, REFSQ Workshops. pp. 62–68.
Karlsen, I.K., Maiden, N., Kerne, A., 2009. Inventing requirements with creativity support tools. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 162–174.
Kreft, I.G., De Leeuw, J., 1998. Introducing Multilevel Modeling. Sage.
Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents. In: International Conference on Machine Learning. pp. 1188–1196.
Lefons, E., Pazienza, M., Silvestri, A., Tangorra, F., Corfiati, L., De Giacomo, P., 1977. An algebraic model for systems of psychically interacting subjects. IFAC Proc. Vol. 10 (12), 155–163.
Lemos, J., Alves, C., Duboc, L., Rodrigues, G.N., 2012. A systematic mapping study on creativity in requirements engineering. In: Proceedings of the Annual ACM Symposium on Applied Computing, SAC, pp. 1083–1088.
Linstead, E., Lopes, C., Baldi, P., 2008. An application of latent dirichlet allocation to analyzing software evolution. In: Proceedings of the International Conference on Machine Learning and Applications, ICMLA. pp. 813–818.
Lloyd, S., 1982. Least squares quantization in PCM. IEEE Trans. Inf. Theory 28 (2), 129–137.
Maiden, N., Gizikis, A., Robertson, S., 2004a. Provoking creativity: Imagine what your requirements could be like. IEEE Softw. 21 (5), 68–75.
Maiden, N., Jones, S., Karlsen, I.K., Neill, R., Zachos, K., Milne, A., 2010. Requirements engineering as creative problem solving: A research agenda for idea finding. RE 57–66.
Maiden, N., Manning, S., Robertson, S., Greenwood, J., 2004. Integrating creativity workshops into structured requirements processes. In: Proceedings of the ACM Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques. pp. 113–122.
Maiden, N., Ncube, C., Robertson, S., 2007. Can requirements be creative? Experiences with an enhanced air space management system. In: Proceedings of the International Conference on Software Engineering, ICSE. pp. 632–641.
Maiden, N., Robertson, S., 2005. Integrating creativity into requirements processes: Experiences with an air traffic management system. In: Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on. IEEE, pp. 105–114.
Mann, H.B., Whitney, D.R., 1947. On a test of whether one of two random variables is stochastically larger than the other. Ann. Math. Stat. 50–60.
Marcus, M.P., Marcinkiewicz, M.A., Santorini, B., 1993. Building a large annotated corpus of English: The Penn Treebank. Comput. Linguist. 19 (2), 313–330.
Mavin, A., Wilkinson, P., Harwood, A., Novak, M., 2009. Easy approach to requirements syntax (EARS). In: Requirements Engineering Conference, 2009. RE'09. 17th IEEE International. IEEE, pp. 317–322.
Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119.
Mumford, M.D., 2003. Taking stock in taking stock. Creat. Res. J. 15 (2–3), 147–151.
Murukannaiah, P.K., Ajmeri, N., Singh, M.P., 2016. Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in Crowd RE. In: Requirements Engineering Conference (RE), 2016 IEEE 24th International. IEEE, pp. 176–185.
Nuseibeh, B., Easterbrook, S., 2000. Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, ICSE 2000. pp. 35–46.
O'Quin, K., Besemer, S.P., 1989. The development, reliability, and validity of the revised creative product semantic scale. Creat. Res. J. 2 (4), 267–278.
Parkhurst, H.B., 1999. Confusion, lack of consensus, and the definition of creativity as a construct. J. Creat. Behav. 33 (1), 1–21.
Plucker, J.A., Beghetto, R.A., Dow, G.T., 2004. Why isn't creativity more important to educational psychologists? Potentials, pitfalls, and future directions in creativity research. Educ. Psychol. 39 (2), 83–96.
Pohl, K., 2010. Requirements Engineering: Fundamentals, Principles, and Techniques. Springer Publishing Company, Incorporated.
R Core Team, 2018. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL https://www.R-project.org/.
Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. 20, 53–65.
Runco, M.A., Jaeger, G.J., 2012. The standard definition of creativity. Creat. Res. J. 24 (1), 92–96.
Sakhnini, V., Berry, D.M., Mich, L., 2010. Validation of the effectiveness of an optimized EPMcreate as an aid for creative requirements elicitation. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, pp. 91–105.
Sakhnini, V., Mich, L., Berry, D.M., 2012. The effectiveness of an optimized EPMcreate as a creativity enhancement technique for Web site requirements elicitation. Requir. Eng. 17 (3), 171–186.
Searle, S.R., Casella, G., McCulloch, C.E., 2009. Variance Components, Vol. 391. John Wiley & Sons.
Snijders, T.A., 2011. Multilevel Analysis. Springer.
Stein, M.I., 1953. Creativity and culture. J. Psychol. 36 (2), 311–322.
Sternberg, R.J., Sternberg, R.J., 1999. Handbook of Creativity. Cambridge University Press.
Tricentis, 2019. Software fail watch: 5th edition. https://goo.gl/WzXcBe, (Accessed 23 March 2019).
White, A., Shen, F., Smith, B.L., 2002. Judging advertising creativity using the creative product semantic scale. J. Creat. Behav. 36 (4), 241–253.
Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biom. Bull. 1 (6), 80–83.
Winter, B., 2013. Linear models and linear mixed effects models in R with linguistic applications. arXiv preprint arXiv:1308.5499.
Yin, R.K., 2013. Case Study Research: Design and Methods. Sage Publications.
Zachos, K., Maiden, N., 2008. Inventing requirements from software: An empirical investigation with Web services. In: Proceedings of the International Requirements Engineering Conference, RE. pp. 145–154.
Zhang, T., Ramakrishnan, R., Livny, M., 1996. BIRCH: an efficient data clustering method for very large databases. In: ACM Sigmod Record, Vol. 25. ACM, pp. 103–114.

Dr. Do has recently completed his Ph.D. in the Department of Computer Science and Engineering at Mississippi State University. He received his undergraduate degree from Vietnam National University, Hanoi. His research interests are requirements engineering (RE) and creativity in software requirements.

Dr. Bhowmik completed his Master and Ph.D. in the Department of Computer Science and Engineering at Mississippi State University. He is currently an Assistant Professor at the same department. His research interests include creativity in requirements engineering (RE) and addressing software security in RE.

Dr. Bradshaw completed his Ph.D. from Carnegie Mellon University. He is currently a professor of Psychology at Mississippi State University. His research interests include web-based education, individual differences, decision-making and errors, complex problem solving, and psychology of scientific discovery and invention.