The Role of Environment Assertions in Requirements-Based Testing

Tanmay Bhowmik*, Surendra Raju Chekuri*, Anh Quoc Do*, Wentao Wang†, and Nan Niu†
* Department of Computer Science and Engineering, Mississippi State University, USA
† Department of Electrical Engineering & Computer Science, University of Cincinnati, USA
tbhowmik@cse.msstate.edu, src463@msstate.edu, aqd14@msstate.edu, wang2wt@mail.uc.edu, nan.niu@uc.edu

***Abstract*—Software developers dedicate a major portion of their development effort towards testing and quality assurance (QA) activities, especially during and around the implementation phase. Nevertheless, we continue to see an alarmingly increasing trend in the cost and consequences of software failure. In an attempt to mitigate such loss and address software issues at a much earlier stage, researchers have recently emphasized on the successful coordination of requirements engineering and testing. In addition, the notion of requirements-based testing (RBT) has also emerged with a focus on checking the correctness, completeness, unambiguity, and logical consistency of requirements. One seminal work points out that requirements reside in *the environment* which is comprised of certain problem domain phenomena. Environmental assertions, which connect some of these phenomena in the indicative mood, play a key role in deciding whether a software solution is acceptable. Despite that requirements are located in the environment, little is known about if and how the environment assertions would impact testing and QA activities. In order to address this gap, we present a detailed empirical study, with 114 developers, on the prominence of environment assertions in RBT. Although the results suggest that paying attention to correct, complete, and useful environment assertions has a positive impact on RBT, developers often face difficulty in formulating good assertions from scratch. Our work, to that end, illuminates the potential usefulness of automated support in generating environment assertions.**

***Index Terms*—environment assertions; requirements-based testing; requirements engineering and testing; requirements engineering**

I. Introduction

As software systems have become ubiquitous in the modern society, people's constant reliance on them in daily life requires these systems to meet expectations of continuously improved quality and reliability [1]. In order to deliver quality software, developers dedicate a major portion of their development effort towards testing and quality assurance (QA) activities, especially during and around the implementation phase of the software development life cycle (SDLC) [2]. Nevertheless, we continue to see an alarmingly increasing trend in the cost and consequences of software failure. According to the fifth edition of Software Fail Watch by Tricentis [3], software failures cost our economy US$1.7 trillion in 2017 (up from US$1.1 trillion in 2016), affecting 3.6 billion people and causing more than 268 years in downtime. Such numbers are an extraordinary reminder of the necessity of improved software testing in every industry and the far-reaching impacts of software failure [3].

There is a consensus in the software community that faults discovered later in SDLC have higher negative impacts and are more expensive to fix than those discovered early [4]. In an attempt to address software issues at a much earlier stage, researchers have recently emphasized on the successful coordination of requirements engineering and testing [5], [6]. Along this line, studies propose model-based techniques to verify and validate requirements [7], [8], investigate automated support to generate test cases from requirements [9]–[13], and suggest approaches to keep requirements and test documents aligned [14], [15]. In addition, the notion of requirements-based testing (RBT) [16] has also emerged, which is a type of black-box testing [1] with a focus on checking the correctness, completeness, unambiguity, and logical consistency of requirements [17], [18].

In his foundational work, Jackson conceptualizes the notion of *the environment* and *the machine* with certain phenomena and indicates that requirements are located in the environment [19]. This environment is characterized by certain assertions comprised of conditions or properties over the phenomena in the environment. As Jackson points out — a large number of software issues stem from faulty reasoning or approximations about the environment [19]. In other words, limited or no consideration of environment assertions may cause an inadequate analysis of the task in hand, leading to various software issues. Although prior research introduced a formal requirements model for the software black-box behavior along with the assertions in a complex process control system [20], the objectives were writing requirements specifications to be readable and reviewable by application experts. In other words, the investment in constructing one state-based model that includes all of the needed information and no more is considered to pay off in terms of discovering more requirements-level errors [20]. However, little is known whether a less perfect domain model impacts the discovery of software errors.

In order to address this gap, we present a detailed empirical investigation on the prominence of environment assertions in requirements engineering and testing. In particular, we examine how environment assertions influence developers' RBT activities and outcomes of such a testing process. To that end, we conduct an empirical study with 114 developers performing RBT on iTrust [21], a Java-based medical records software system, where the experimental group and the control

2332-6441/19/$31.00 ©2019 IEEE
DOI 10.1109/RE.2019.00019

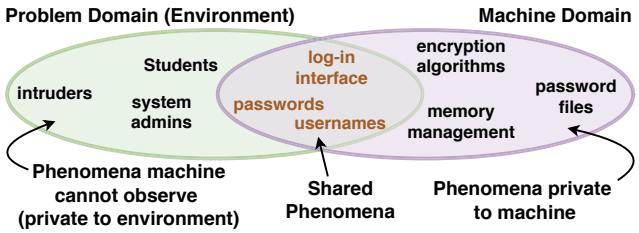

Fig. 1: The environment and the machine for a hypothetical authentication scenario (adapted from [22]).

group carry out their activities with and without explicitly considering environment assertions, respectively. The results suggest that paying attention to quality environment assertions, i.e., assertions that are correct, complete, and useful, has a positive impact on RBT in terms of the quality of test cases and test outcomes. However, our study also reveals that developers often face difficulty in formulating good assertions from scratch. In this paper, unless otherwise mentioned, we use the term developers to indicate stakeholders participating in the development process and conducting testing activities.

The contributions of this work lie in carrying out an empirical study on the significance of Jackson's [19] conceptualization of the environment and the machine in the context of testing and QA activities. In addition, we demonstrate a practical application of environment assertions in RBT and illuminate the potential usefulness of an automated support to help developers generate environment assertions. In what follows, we present background information on the meaning of requirements and RBT in Section II. We then detail the empirical study and analyze the results in Section III. Further explanation of the results and some additional insights from our study are presented in Section IV. Section V discusses validity threats to the work, and finally, Section VI concludes the paper.

II. Background and Related Work

A. The Environment and the Machine

Jackson first conceptualized the software world using the notion of the environment and the machine defined by certain phenomena [19]. In one of the foundational papers in RE [22], he teased out the meaning of requirements by distinguishing two domains: the problem domain, later denoted as the environment [19], and the machine domain. Understanding the problem domain helps to identify constraints that define the range of conditions within which the system may operate. The i^* framework [23], for example, supports problem domain modeling by focusing on the discovery of the system actors' intentionality in terms of their goals and the delegation of responsibility between actors that permit their goals to be fulfilled. Using i^*, the "as-is" business processes and the roles, goals, and priorities of the actors within the organization can be modeled. Such domain models form the foundation from which alternative solutions can be explored and the system requirements can be formulated.

Jackson points out that the environment is characterized by certain assertions comprised of conditions or properties over the phenomena in the environment, whereas the machine domain is private to the intended software and the computing devices in which the software operates [22]. To that end, as Jackson defines, environment assertions are phenomena in the environment that are true whether or not we ever build the proposed system [19], and requirements need to be expressed in terms of relationships among such environment assertions [22].

Fig. 1 demonstrates Jackson's conceptualization through a hypothetical "authorized access to lab machines" scenario. In this context, some environment phenomena, including students and intruders, are exclusively private to the environment. Similarly, phenomena such as encryption algorithms and password files are private to the machine. Besides, some phenomena, e.g., usernames and passwords, are shared with and observable to the machine. According to Jackson [19], [22], both requirements (R) and environment assertions ($\mathcal{E}$) reside in the environment where the former "express conditions over the phenomena of the environment that we wish to make true by installing the machine", and the latter "express the conditions over the phenomena of the environment that we know to be true irrespective of the properties and behavior of the machine". Jackson [22] further indicates that the specifications (S) are restricted requirements expressed solely based on the shared phenomena. Building upon such notions of R, $\mathcal{E}$, and S, Jackson formulates the famous entailment relationship in RE [19]:

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

In other words, if the environment holds the assertions we claim, and the machine behaves according to the specifications, then satisfaction of the requirements, i.e., the software behaving in a correct and expected manner, can be carried out.

In sum, Jackson pointed out that requirements are, in fact, conditions over the right events and states of the environment and the correctness of software depends on correctness of the environment which can ultimately be established through the accuracy and completeness of environment assertions [19]. For example, in Fig. 1, a requirement R may be: "allow only active students to log into the lab computer". According to Jackson [19], R is in the optative mood, expressing a wish. The environment assertion $\mathcal{E}$, on the other hand, is in the indicative mood, expressing what is claimed to be a known truth. Here, a relevant $\mathcal{E}$ may be: "a teaching assistant is an active student." Given this $\mathcal{E}$ and the specification S of "correct <username, password> pair leads to a successful log in", a teaching assistant shall be granted access to the lab computer.

In the above example, other environment assertions may be relevant, e.g., "a person would not share his or her <username, password> with others", "system admin is not a malicious user", etc. In practice, identifying domain terms such as "student", "teaching assistant", and "<username, password>" can

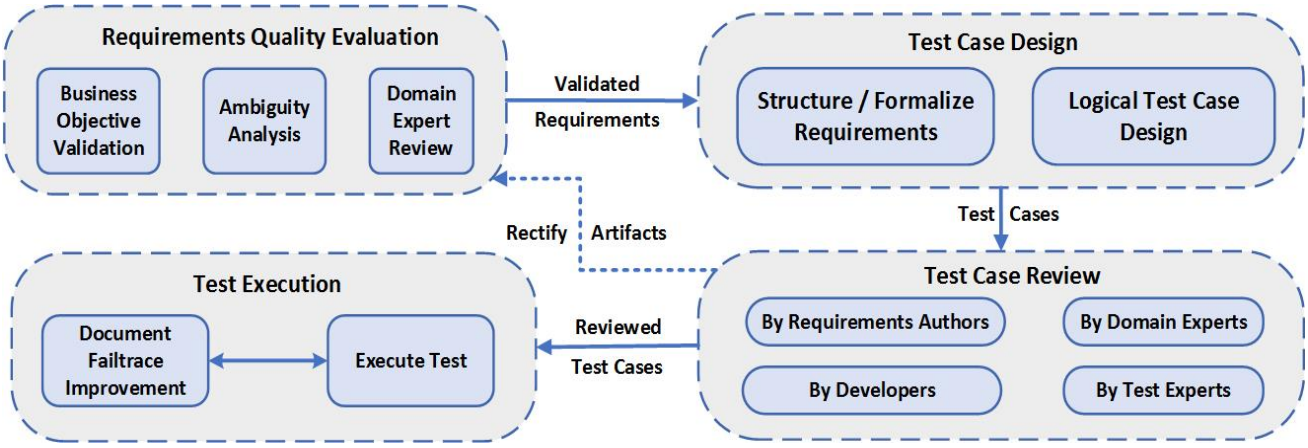

Fig. 2: The RBT process flow (adapted from [18]).

help the formulation of environment assertions, and Sawyer *et al.* [24] showed how combinations of lexical and shallow semantic analysis techniques could automatically recognize domain terms and nondomain terms.

Our research shares the objective of Leveson and her colleagues [20], [25], [26] to construct an accurate and complete set of environment assertions. Different from using a state-based formal modeling approach [20], [25], [26], we made no initial investment of developing environment assertions because we are interested in applications not only in safety-critical domains. Unlike the objective of supporting application experts (e.g., airframe manufacturers) to discover requirements-level errors by Leveson *et al.* [20], [25], [26], our goal is to support developers and testers to detect software errors stemmed from faulty or incomplete environment assertions.

B. Requirements-Based Testing (RBT)

In order to deliver high quality software on time, research in recent years has stressed on proper alignment of requirements engineering and testing activities [6], and has proposed several avenues to attain such alignment. For example, Aoki and Matsuuara [7] have introduced a method, leveraging model checking and Common Criteria security knowledge, to *verify* security requirements specified in Unified Modeling Language (UML). Following the growing complexity of model-based requirements validation for real-time systems, Zhou *et al.* [8] have suggested an observer-based lightweight technique to *validate* both functional and non-functional requirements written in a formal specification language.

In order to support testing activities based on requirements, researchers have also proposed several tools and techniques. Almohammad *et al.* [9] have presented an automated tool to generate test cases from requirements for different coverage criteria. Freudenstein and colleagues [10] have developed Specmate to partly automate the design of test cases from requirements. Singi *et al.* [11] have advocated the notion of visual requirements for digital applications and presented a graph-model based approach to automatically generate test cases from those requirements. Further automated supports are also proposed for activities, such as abstract test case generation from requirements model [12] and development of model-based test cases from requirements descriptions [13], [27].

In an attempt to keep requirements and test documents aligned, researchers have also introduced several approaches, such as automated generation of test guidance [14], and the roadmap view for quality requirements and test results [15]. In the literature, we also identify the emerging notion of RBT, which, in our opinion, can promote further alignment between RE and testing activities. As indicated by Skoković and Skoković [18], the RBT process addresses two major aspects: validation of requirements, and designing a necessary and sufficient set of test cases from a black-box perspective. Unlike traditional testing activities, RBT does not assume that the requirements in hand are correct and complete. Rather, it drives out ambiguity and drives down the level of details through a process involving four major sets of activities (cf. Fig. 2) [28], which are discussed next.

Requirements quality evaluation: This is one of the core components of RBT that stands out from other traditional testing techniques. The activities along this line include validation of requirements against business objectives [18], i.e., evaluating how useful a requirement is for the intended system and to what extent it is adaptive to task constraints. In addition, assuming a domain expert's role, the test engineers need to conduct an initial ambiguity review and domain expert review to verify the correctness, completeness, ambiguity, and logical consistency of the requirements. Let "the lab machine shall be accessible only to authorized personnel" be a requirement in our hypothetical scenario (cf. Section II-A). Quality evaluation activities, especially in light of environment assertions, may unveil potential issues, e.g., ambiguity with the phrase "authorized personnel", and help clarify expected system behavior for

less obvious personnel, including custodians and electricians.

Test case design: This includes structuring/formalizing requirements and designing logical test cases. The former involves expressing a requirement as a flow of activities that naturally depicts precedence dependency between actions. The latter indicates writing logical steps for the test that captures granular level functionality of the requirement. In our continuing example, "access to the lab machine shall be granted only after the user provides valid credentials, e.g., valid username and password pair" could be considered as a restructured requirement. On the other hand, a logical test case could be: identify a couple of valid and invalid credentials, e.g., username, password pairs $\rightarrow$ type in credentials, both valid and invalid $\rightarrow$ if the credential is valid, access granted; if invalid, access denied with an error message.

Test case review: In order to further assure quality, the test cases are reviewed by different stakeholders, including domain experts, requirements authors, developers, and test experts. Here, the idea is to conduct review from these different perspectives, not necessarily by different personnel, in order to identify and rectify any inconsistency between the test cases and respective artifacts developed by different stakeholders.

Test execution: Lastly, the activities for this part involve execution of test cases following the logical steps teased out during test case design and documenting the test outcomes, i.e., the execution passes or fails the test with trace information to reproduce failure, if any. In addition, RBT also emphasizes on describing further improvement of the requirement with justification, if applicable.

It should be noted that current literature does not identify the importance of environment assertions for RBT activities in an explicit manner. In our opinion, however, such assertions should be crucial in this context as they constitute the much needed knowledge for domain experts and other stakeholders to carry out quality evaluations. An objective of our work, to that end, is to examine this assumption.

III. Empirical Study

A. Research Questions

A major objective of this work is to empirically investigate Jackson's indication that limited or no consideration of environment assertions may lead to inadequate development activities, thereby resulting in software issues [19]. To that end, we pick RBT as a development activity and investigate:

> ***Central research question*:** How do environment assertions influence developers' RBT activities and the outcomes of such a process?

The reasons behind considering RBT include: i) It is a black-box testing technique, thereby does not require knowledge about the internal structure of the source code, and ii) RBT's focus on improving requirements quality provides an opportunity to address software issues at an earlier stage making it aligned with the philosophy of requirements engineering and testing. Although RBT involves four major sets of activities (cf. Section II-B), in order to keep our study within a manageable scale, we particularly focus on two of those kinds, namely, test case design and test execution. The rationale behind this choice is threefold. First, in this study, we choose requirements from a real-world and mature open-source software (OSS) system named iTrust [21], and expect that its requirements and use cases are already well-defined. Second, given the limited availability of original requirements' authors and programmers, it would be impractical to incorporate test case review by such stakeholders. Finally, we posit, test case design and execution activities, especially test outcome analysis and further improvement recommendations for requirements, demand some skills of a domain expert, thereby providing us a clear idea about environment assertions' role in RBT.

Based on the rationale stated above, we first want to examine if the consideration of environment assertions has any impact on the quality of test cases designed during RBT activities. To that end, we ask the research question: RQ_1 – ***Do environment assertions lead to better test cases during RBT activities?***

As indicated earlier, we are also interested in exploring the role of environment assertions in test execution outcomes, in particular, on the likelihood of recommending further improvements for requirements. Thus our second research question is: RQ_2 – ***Do environment assertions lead to a higher likelihood of suggesting improvements for requirements?***

Furthermore, if environment assertions indeed have an impact on the RBT activities, it is logical to speculate that the quality of the assertions may play a part in the overall execution outcomes. Thereby, we ask the following research question: RQ_3 – ***How does the quality of environment assertions influence the quality of suggested improvements?*** In what follows, we discuss our subject system and study setup to address these research questions.

B. Subject System

As mentioned earlier, we consider iTrust as the subject system in this study. It is a security-critical Java medical records software system [21], which is an open source application originally developed by the software engineering students at North Carolina State University. It provides patients with a means to keep up with their medical records history and to communicate with their doctors [21]. Over several years, iTrust has been a subject system of numerous software engineering research efforts [29]–[31] providing critical insights on important topics including traceability and vulnerability discovery. In our study, we use version 23 of iTrust which includes 112,987 lines of code and 784 Java source files.

C. Study Setup

In this study, we need developers to have some knowledge regarding environment assertions and, to some extent, RBT. As these topics are not usually covered, to a considerable extent, by any regular software engineering course to our knowledge, we suspect that regular software developers, in general, may not be familiar with these ideas in an explicite manner. To that end, our study setup involves three major phases, including training potential participants, conducting experiment, and

evaluating participants' work-products. In what follows, we detail our activities along these lines.

Training potential participants: As our potential study participants, we consider students enrolled in a split-level (i.e., a course that includes both graduate and undergraduate students) software engineering course at two different universities in North America. In order to train the students with the concepts of environment assertions and RBT, the instructors dedicate two 75-minute classes, one for each topic. These classes are delivered in a week towards the beginning of a semester, each containing a 30- to 35-minute lecture on the theoretical concepts with the remaining spent on in-class exercise and practice-problems. In order to facilitate further practice, the students individually work on an additional assignment, designed on these topics, in a lab class in the following week.

To maintain consistency, the two instructors coordinate with each other very closely and run their respective classes in a synchronized manner. They deliver the same lecture materials, same in-class exercise and practice-problems, and the same lab-assignments to make sure that the students develop an anticipated minimum level of proficiency in these topics. In an attempt to ensure that the potential participants are familiar with medical records software systems, during another week later in the semester, each of them researches this domain online and individually submits a domain analysis report.

Experimental design: In this work, we follow a randomized two-group posttest-only design [32]. During a week towards the end of the semester, we recruit 114 participants, includes 31 graduate students (both MS and PhD admits) and 83 senior-level undergraduate students, from the courses stated above. These participants are either from computer science or software engineering major, with some majoring in both, and with a median of 3.5 years of software development experience. The experience we count here includes 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 form an experimental group (G_{exp}) and a control group (G_{cnt}) by randomly assigning the recruits, where each group has 57 participants. Note that we also make confidentiality agreement with the participants and, after the semester officially ends, we obtain their consent about using the work for scientific research.

During the study, each participant is provided with a document that contains the title, description, and detailed use case of an iTrust requirement "Schedule Appointments" (cf. Fig. 3). As the use case suggests, this requirement can be decomposed into several smaller units, such as patient requesting appointment and LHCP reviewing appointment request. The participants are asked to break down the requirement into a number of smaller units they find appropriate and work on as many units as possible in 1 hour. Each participant works individually on a lab machine that has iTrust deployed using Eclipse IDE. For each unit, a participant in the experimental group writes a brief description of the unit in concern, brainstorms and notes down some environment assertions relevant to the unit, writes the test cases, executes them by running iTrust, and writes the test outcomes and possible improvements for the unit (if she can think of any). A control group participant, on the other hand, conducts all these activities except she is not instructed to think of environment assertions at any stage. The participants are also asked to justify their test outcomes. After all the participants turn in their documents, we follow a double-blind peer review strategy to evaluate their work, which is discussed next.

Requirement: Schedule Appointments

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

An LHCP views a list of pending appointment requests. Each appointment request is listed as being pending, approved, or rejected. The LHCP is presented with an option to approve or reject each pending appointment request. When the LHCP approves or rejects an appointment request, a message is sent to the Patient indicating the request status. When the date of an appointment request has passed, it is no longer displayed.

Fig. 3: "Schedule Appointments" use case.

Evaluation: We randomly assign each participant two documents created by two other participants in the same group for a double-blind review. Note that our participants received training on different review techniques in a prerequisite software engineering course. The participants evaluate the quality of the improvement recommendations, as well as the environment assertions in case of the experimental group, for **clarity**–unambiguous and provides an appropriate level of detail [33], **correctness**–accurately reflects conventional knowledge and a good fit for the system, and **usefulness**–provide value or utility to the requirement [33]. In doing so, they assign ratings at a 5-point Likert scale: 1=very low, 2=low, 3=medium, 4=high, 5=very high and provide a justification of the ratings. At the end, the experimental group participants share their thoughts on RBT with environment assertions.

D. Results and Analysis

RQ_1 – **Do environment assertions lead to better test cases during RBT activities?** As the main objective of software testing is to uncover defects [34], we operationalize the notion of a "better test case" as the one that ultimately

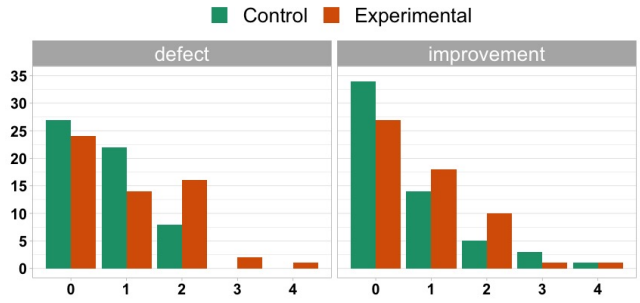

Fig. 4: Participants identifying defects and improvements.

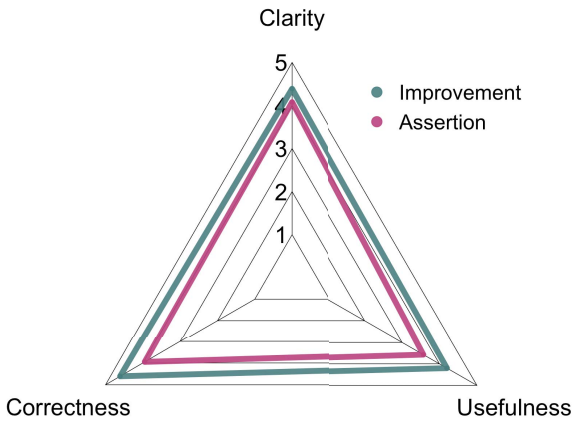

Fig. 5: Quality ratings: Improvements Vs Assertions.

uncovers more defects for the system being tested. In other words, while testing a certain functionality *f*, a test case T_1 is better than test case T_2 if f fails T_1 but not T_2. Accordingly, in order to answer RQ_1, we examine the test cases written by each participant in G_{exp} and G_{cnt} and count the cases that iTrust ultimately fails, i.e., the number of defects detected by each participant. Fig. 4 summarizes our findings along this line. On an average, a participant in G_{exp} reports 0.98 defects (ranging from 0 to 4), which is higher than that of a participant in G_{cnt} (avg. 0.67, ranging from 0 to 2). In G_{exp}, 33 (i.e., 57.9%) participants identify one or more defects which is higher than G_{cnt} (30 or 52.6% participants detecting 1 or more). In addition, 19 participants in G_{exp} find two or more defects whereas only 8 participants in G_{cnt} find those many defects.

We further conduct Mann-Whitney-Wilcoxon test [35], [36], which is a non-parametric equivalent of the t-test [37], to statistically examine the difference between our two groups along this line. We use R [38], a popular software package for statistical computing, to perform this test. We find a *W* value of 1374 with *p-value* = 0.1287, which apparently does not indicate any statistically significant difference between G_{exp} and G_{cnt} at $\alpha = 0.05$. In sum, although the earlier stated descriptive statistics provide evidence, to some degree, that *assertions may lead to better test cases during RBT activities*, sophisticated statistical tests yet do not show a significant difference. The discussion section sheds some light along this line. It is worth mentioning that some of the control group participants identify a single common defect about appointment request. Further post-hoc analysis uncovers additional explanation of this finding in terms of environment assertions, which is discussed in Section IV.

RQ_2 – **Do environment assertions lead to a higher likelihood of suggesting improvements for requirements?** In order to answer this research question, we shift our attention to the number of improvements proposed by each participant in both G_{exp} and G_{cnt}. We find that a G_{exp} participant, on an average, makes 0.79 improvement recommendations for the requirement "Schedule Appointments", whereas, in case of a G_{cnt} participant, the average is lower (0.65). For both the groups, the number of recommendations from a participant ranges from 0 to 4. As many as 30 (i.e., 52.6%) participants in G_{exp} make one or more improvement recommendations, which is higher compared to G_{cnt} where only 23 (or 40.4%) participants make any recommendation.

A Mann-Whitney-Wilcoxon test [35], [36] on the number of improvement recommendations by the participants in each group barely suggests a statistically significant difference between G_{exp} and G_{cnt} at $\alpha = 0.05$ (W = 1436, *p-value* = 0.09414). Similar to what we have noticed for RQ_1, here, again we find descriptive statistics favorable for an implication that *an explicit consideration of environment assertions during RBT may lead to a higher likelihood of recommending improvements*. Mann-Whitney-Wilcoxon test provides a promising *p-value* (below 0.1), however, it narrowly falls short of exhibiting significant statistical evidence at α=0.05 level of significance. Section IV provides additional insights along this line.

RQ_3 – **How does the quality of environment assertions influence the quality of suggested improvements?** In order to answer this research question, we need to analyze both the quality of the environment assertions and the suggested improvements and examine if there exists an association between these qualities. To that end, we analyze the data collected during the evaluation phase (cf. Section III-C), in particular the peer-review ratings for the test documents from G_{exp} that include specific improvement suggestions (30 out of 57, i.e. 53% G_{exp} participants recommended improvements). Our objective is to investigate if there is a correlation between the quality ratings (i.e. correctness, completeness, and usefulness) for environment assertions and corresponding improvement suggestions made by a participant. For each of such test documents, we obtain the average ratings for the assertions and suggestions and calculate Spearman's rank-order correlation, also known as Spearman's ρ, which is a nonparametric measure of the strength and direction of association between two variables measured on at least an ordinal scale [37]. The results suggest a statistically significant positive correlation between the ratings ($\rho = 0.54$, $p < 0.00001$) with moderate strength [39].

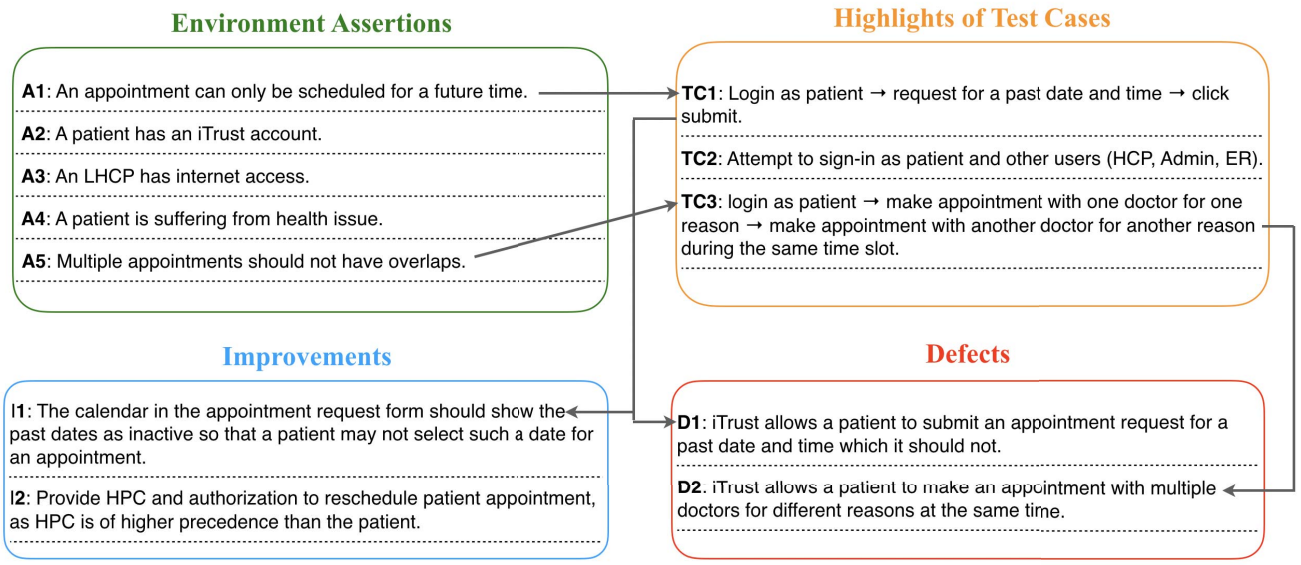

Fig. 6: Sample outcomes of the participants' RBT activities.

Fig. 5 provides a radar chart presenting the relation between quality attributes correctness, completeness, and usefulness for assertions and improvement suggestions. We notice that the granular level attributes also display positive associations. For example, if the average rating for correctness of environment assertions is high, so is for the corresponding improvement suggestions. Ratings for completeness and usefulness also exhibit similar trends. Such observations further substantiate the result from Spearman's correlation analysis. In other words, getting the average of these attributes apparently does not lead to losing valuable information. Based on these findings, we conclude that *the quality of environment assertions positively influences the quality of suggested improvements*. That is, a developer with better environment assertions in her disposal is more likely to make correct, complete, and useful improvement suggestions.

Based on the results and analysis presented so far, we notice that our empirical study provides some preliminary evidence supporting the hypothesis: "high quality" environment assertions indeed positively influence developers' RBT activities and the outcomes of such a process. Although we lack enough statistical evidence in some cases, we expect that a closer look into the collected data will provide further explanation of our results and help us gain additional insights about the role of environment assertions in RBT activities.

IV. Discussion

Fig. 6 presents some sample outcomes of the RBT activities conducted by our participants. As mentioned in the preceding section, apart from Likert scale ratings, our participants also provide explanations to justify their ratings and share their opinions on the use of environment assertions in RBT. In order to obtain some additional insights about the initial results of our research questions, we conduct further analysis on such qualitative data. In this section, we detail some of our findings along with supporting anecdotal evidence from the post-hoc analysis where part of the discussion is anchored around the information presented in Fig. 6.

A. Implicit Environment Assertions Play a Part

As we pointed out earlier, the majority of the participants commonly detected a defect with the implementation of iTrust's appointment request, which is:

> iTrust allows a patient to submit an appointment request for a past date and time which it should not.

In fact, 22 (i.e., about 39%) G_{cnt} participants uncovered a single defect, which is predominantly the one mentioned above. After examining the explanations written by the participants, we obtain an interesting insight along this line. Although the wording of the explanation varies from participant to participant, the common theme we observe is as follows.

> Scheduling an appointment by design implies that we are going to set an appointment for a future time. It is one of the basics of scheduling appointment. iTrust allows the patient to successfully submit a request for a past date and time.

In other words, the participants indicated that whenever we try to schedule an appointment with someone, regardless the purpose, a "valid time" for the actual appointment should be in the "future", i.e., sometime after the scheduling activity has taken place. Thereby, a basic condition over the phenomena in the problem domain (i.e., the environment) of an appointment scheduling scenario would be, "An appointment can only be scheduled for a future time." This is, by definition, an environment assertion, since it remains true even if we never build iTrust or any other system of its kind. Note that not all participants detected this defect. Those who did, actually wrote

this as an environment assertion (in case of G_{exp}) or provided the above explanation (mostly in case of G_{cnt}). Those who did not, apparently took the correctness along this course as granted, only included test cases with future dates and times, and largely focused on the "conflicting schedule" scenario.

Although not as prevalent as the aforementioned "appointment at a past date and time" issue, other relatively common defects with the appointment request feature mentioned by our participants include:

> iTrust allows a patient to make appointments with multiple doctors for different reasons at the same time.

In case of the G_{exp} participants who identified this defect, we note that they mentioned an environment assertion that mostly reads, "multiple appointments should not have overlaps". Out of the 8 G_{cnt} participants who detected more than 1 issue, 5 explicitly included this in their list of defects and gave some additional explanations. One of those participants provide the following argument.

> Someone can not physically meet multiple people in different places at the same time. I did hear about something called a medical board where multiple doctors work together to handle a critical patient. I do not think that is a kind of appointment a patient herself makes using a system like iTrust. I believe the critical care unit or emergency room doctors form such a board as needed.

Clearly, the theme of this explanation falls in line with the assertion mentioned in the preceding paragraph. Similar to the "appointment at a past date and time" case discussed earlier, the participants who did not pay attention to the time overlap related issues of scheduling and appointment missed to incorporate this aspect into their test cases.

This analysis helps us gain valuable insights about environment assertions and testing. First, explicit consideration of environment assertions may help us avoid subtle mistakes while designing relevant test cases. Second, even if we do not concretely spell out an environment assertion, the knowledge about it residing in our minds may still play a part in doing the tests right (as it happened for those G_{cnt} participants). In this paper, we name this "implicit environment assertion".

B. The Quality of Environment Assertions Matters

Our data suggest that an explicit consideration of environment assertions alone may not be sufficient for a developer to obtain effective outcomes from RBT activities. This is indeed an implication of the results we obtained for RQ_3 suggesting better quality environment assertions lead to better RBT outcomes. Consequently, low quality assertions, i.e., incorrect, incomplete, or not directly relevant to the requirement under consideration, may not have a positive effect on the testing activities and outcomes. In what follows, we substantiate this observation through some examples from our study.

In Fig. 6, we observe some low quality environment assertions, such as "A patient has an iTrust account", "An LHCP has

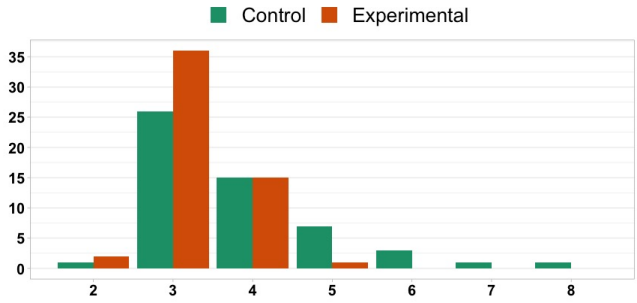

Fig. 7: Number of units completed: Control Vs Experimental.

internet access", "A patient suffers from health issues", etc., that are not necessarily incorrect or incomplete, but are of little direct use for testing a schedule appointment functionality. Such assertions may often cause distraction in RBT activities and, the process may not produce any useful outcome at the end. As the following justification for the assertion ratings from a peer-reviewer points out:

> I am rating these assertions very low. Don't know how they can be relevant to appointment request/approval functionality. The document spends much time designing test cases for patient/LHCP log-in activities. Those are basic user activities irrespective of what functionality you use. No surprise these activities do not identify a critical defect with request appointment.

Quality assertions, e.g., "An appointment can only be scheduled for a future time", be that explicitly written or implicit in the developer's mind, apparently lead to appropriate test cases. They help detect critical defects, such as "The system allows an appointment request to be submitted for a past date and/or time", and further guide improvement recommendations, including "The calendar in the appointment request form should show the past dates as inactive so that a patient may not select such a date for an appointment" (cf. Fig. 6). For similar reasons, although the results for RQ_2 indicate higher likelihood of suggesting improvements when assertions are under specific consideration, a few suggestions from G_{exp} received slightly lower quality ratings (avg. 4.22) compared to G_{cnt} (avg. 4.29). These findings suggest that not just the assertions but their quality matters. In our opinion, this is one of the reasons behind the absence of sufficient statistical evidence for the answers of the first two research questions (cf. Section III-D).

C. Writing Assertions from Scratch is Challenging and Time Consuming

One important aspect we should reiterate is that a G_{exp} participant explicitly wrote down some environment assertions, whereas a G_{cnt} participant did not. Formulating environment assertions should require some critical thinking, call for knowledge about the requirement in concern, and certainly take additional time. In fact, during the evaluation phase of our

study, each G_{exp} participant unequivocally pointed out this tedious nature of writing assertions. In our opinion, this aspect might have played a role behind our results.

As mentioned in Section III-C, the "Schedule Appointments" requirement for iTrust could be broken down into smaller units and each participant had exactly one hour to complete the task. Fig. 7 shows a comparison between the two groups with respect to the number of units the participants tested. We find that G_{cnt} participants complete more units (avg. 3.85, ranging from 2 to 8) compared to G_{exp} (avg. 3.28, ranging from 2 to 5). Since we did not specifically define the scope of a unit and the participants had the liberty to decompose the requirement the way they found appropriate, we note that just a comparison of the number of tested units may not convey the true picture. To that end, we manually analyze each document and identify those with testing activities covering the whole requirement. We notice that 26 G_{cnt} participants (45.6%) covered the complete requirement, whereas the numbers are lower in case of G_{exp} (11, i.e., 19.3% participants covered the full requirement).

During peer evaluation, we also asked the experimental group participants to share their thoughts on writing environment assertions while conducting the RBT activities. The common themes we notice in their responses include: i) writing environment assertions is possibly helpful but it takes time; ii) capturing the assertions from scratch demands critical thinking; and iii) in order to help the testing process, it is important to think of assertions that are aligned with the requirement being tested. The following comment from a G_{exp} participant corroborates our observation.

> I think writing environment assertions is not easy. It takes more time for sure, also some deeper knowledge and understanding of the requirement and the software. I found them helpful to do my tests but it took some energy. You need to keep it right you know... The document I just reviewed did not do a good job on that. The tests are not good either. Wish we had more time. Some pointers to think about the assertions would also help.

Based on these findings, we believe the strictly imposed time constraint might have played a role behind the shortage of statistical evidence in cases of RQ_1 and RQ_2.

Not to our surprise, every G_{exp} participant complained about the time constraint of just 1 hour, given that the requirement contained multiple use case steps. As one of the G_{exp} participants pointed out, "This is not a small use case. Given that we need to write some assertions and then move on with testing, I think 1 hour time is unfair". Furthermore, all these participants unanimously agreed that some additional information to start the environment assertions would help. In fact, 17 participants in the experimental group further speculated the prospect of an automated support along this line. One of those participants made a comment:

> I think some of the information we need for assertions is out there somewhere... maybe in the SRS or in some other documents. Why don't we write some script that goes through them and gives us more clues?

We believe our participant has a point. As Cleland-Huang [40] indicated, modern software engineers have access to documents describing almost every conceivable human knowledge. Such documents come in the form of online product catalogs providing rich feature descriptions, sample requirements, feature models, database schemas, high-level designs [40], current requirements, and stakeholder comments [41], [42], among others. Information contained in those documents is potentially a rich source of knowledge for both existing and novel software systems. In addition, we posit that the combination of lexical and shallow analysis techniques presented by Sawyer *et al.* [24] could be utilized to automatically extract environment phenomena from such textual documents. In sum, we believe, the aforementioned information and techniques could be leveraged to help capture additional knowledge about the requirements and their environment in an automated manner. Inspired by this intriguing idea, in our future work, we plan to concentrate on developing such an automated support.

V. Threats to Validity

Construct validity concerns establishing correct operational measures for the concepts being studied [43]. The main constructs in this work include the quality of assertions, test cases, and test outcomes (e.g., improvement suggestions) in RBT. We train our participants along this line through multiple lectures, assignments, and laboratory activities throughout a semester. In order to capture quality, we follow a well-established Likert scale rating for further granular level attributes: correctness, completeness, and usefulness. In addition, we observe corroborating evidence for our findings uncovered through further qualitative analysis (cf. Section IV). To that end, we believe that our work holds construct validity along this line.

Internal validity establishes the accuracy of conclusions drawn upon cause and effect [32]. We follow a randomized two-group posttest-only design [32], which is common for experiments of this nature. We draw our conclusions based on both qualitative and quantitative analyses augmented with relevant statistical tests. We believe, these approaches provide additional validity to our conclusions. However, a confounding factor we identify as "implicit environment assertions" might have created some bias in our study, especially for RQ_1 and RQ_2, which is discussed in Section IV. Nevertheless, we believe this aspect does not affect the overall findings for our central research question addressed in Section III.

External validity concerns establishing the domain to which a study's finding can be generalized [43]. Although we conduct our study by selecting one software from the health care domain, the meaning of environment assertions, i.e, properties in the environment that are true even if we never build the proposed system [22], is the same for any software domain. Therefore, we are confident about the external validity of our

findings and we expect that a different subject system will lead to *similar conclusions*. A limitation of our work, however, is that we select well implemented (probably due to the maturity of iTrust) requirements for RBT that uncovers limited defects and improvement suggestions. If another requirement of iTrust (presumably a newer one) or a requirement of a less mature system (i.e., a software under development but not released) is used, the results may differ.

Reliability of a study suggests that the operations can be repeated with the same results [43]. Our studies involve student participants which could pose some reliability issues. Research, however, indicates that students and professionals are not that different in performing certain software engineering experiments, including requirements analysis tasks [44]. Therefore, we believe the findings of this research are reliable.

VI. Conclusion

In this paper, we first report an empirical investigation on the significance of Jackson's conceptualization of the environment and the machine [19]. In particular, we examine how environment assertions influence developers' RBT activities and find preliminary evidence that paying attention to quality assertions (that are correct, complete, and useful to the task at hand) has a positive impact on such activities. Our study also uncovers the fact that formulating quality assertions from scratch is often challenging and improper assertions may rather hinder the overall objective of testing and QA activities.

With an aim to address the aforementioned issues, in our future work, we will investigate the possibility of developing an automated support to help capture environment assertions in a comprehensive manner. In particular, we plan to leverage existing software knowledge freely available over online resources, and utilize lexical and semantic analysis techniques to provide additional ideas that can help developers formulate quality assertions. If successful, such a support will assist developers in conducting a variety of software engineering activities, including requirements based testing and domain analysis.

Acknowledgment

Our sincere gratitude to Ms. Mona Assaran for helping us in running the experiment. This research is partially supported by U.S. National Science Foundation (NSF) Award CCF-1350487.

References

[1] J. Tian, *Software quality engineering: testing, quality assurance, and quantifiable improvement*. John Wiley & Sons, 2005.

[2] M. Tuteja and G. Dubey, "A research study on importance of testing and quality assurance in software development life cycle (SDLC) models," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 3, pp. 251–257, 2012.

[3] Tricentis, "Software fail watch: 5th edition," https://goo.gl/WzXcBe, last accessed: March 2019.

[4] B. W. Boehm, *Software engineering economics*. Prentice-hall Englewood Cliffs (NJ), 1981, vol. 197.

[5] "Message from the chairs," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, 2014, pp. iii–iv.

[6] J. Larsson and M. Borg, "Revisiting the challenges in aligning RE and V&V: Experiences from the public sector," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2014, pp. 4–11.

[7] Y. Aoki and S. Matsuura, "Verifying security requirements using model checking technique for uml-based requirements specification," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2014, pp. 18–25.

[8] J. Zhou, Y. Lu, and K. Lundqvist, "The observer-based technique for requirements validation in embedded real-time systems," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2014, pp. 47–54.

[9] A. Almohammad, J. F. Ferreira, A. Mendes, and P. White, "Reqcap: Hierarchical requirements modeling and test generation for industrial control systems," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 351–358.

[10] D. Freudenstein, M. Junker, J. Radduenz, S. Eder, and B. Hauptmann, "Automated test-design from requirements-the specmate tool," in *2018 IEEE/ACM 5th International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2018, pp. 5–8.

[11] K. Singi, V. Kaulgud, and D. Era, "Visual requirements specification and automated test generation for digital applications," in *Proceedings of the Second International Workshop on Requirements Engineering and Testing*, 2015, pp. 37–40.

[12] M. F. Granda, N. Condori-Fernández, T. E. Vos, and O. Pastor, "Towards the automated generation of abstract test cases from requirements models," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2014, pp. 39–46.

[13] V. A. de Santiago Junior and N. L. Vijaykumar, "Generating model-based test cases from natural language requirements for space application software," *Software Quality Journal*, vol. 20, no. 1, pp. 77–143, 2012.

[14] S. Hotomski, E. B. Charrada, and M. Glinz, "Aligning requirements and acceptance tests via automatically generated guidance," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 339–342.

[15] R. B. Svensson and B. Regnell, "Aligning quality requirements and test results with quper's roadmap view for improved high-level decision-making," in *2015 IEEE/ACM 2nd International Workshop on Requirements Engineering and Testing*. IEEE, 2015, pp. 1–4.

[16] J. Bowman, "Requirements based software testing method," Apr. 20 2004, uS Patent 6,725,399.

[17] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, 2006, pp. 25–36.

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

[19] M. Jackson, "The meaning of requirements," *Annals of Software Engineering*, vol. 3, no. 1, pp. 5–21, 1997.

[20] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements specification for process-control systems," *IEEE transactions on software engineering*, no. 9, pp. 684–707, 1994.

[21] iTrust, "iTrust: Role-based healthcare," https://152.46.18.254/doku.php, last accessed: March 2019.

[22] M. Jackson, "Problems and requirements [software development]," in *Proceedings of the International Symposium on Requirements Engineering*. IEEE, 1995, pp. 2–8.

[23] E. S. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. IEEE, 1997, pp. 226–235.

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

[25] M. P. Heimdahl and N. G. Leveson, "Completeness and consistency analysis of state-based requirements," in *1995 17th International Conference on Software Engineering*, 1995, pp. 3–3.

[26] M. P. E. Heimdahl and N. G. Leveson, "Completeness and consistency in hierarchical state-based requirements," *IEEE transactions on Software Engineering*, vol. 22, no. 6, pp. 363–377, 1996.

[27] E. Sarmiento, J. C. S. do Prado Leite, and E. Almentero, "C&l: Generating model based test cases from natural language requirements descriptions," in *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. IEEE, 2014, pp. 32–38.
[28] G. E. Mogyorodi, "What is requirements-based testing?" *Crosstalk: The Journal of Defense Software Engineering*, vol. 16, no. 3, p. 12, 2003.
[29] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, August-September 2015, pp. 248–259.
[30] W. Zogaan, P. Sharma, M. Mirahkorli, and V. Arnaoudova, "Datasets from fifteen years of automated requirements traceability research: Current state, characteristics, and quality," in *Proceedings of the 25th International Requirements Engineering Conference (RE)*. IEEE, Sept 2017, pp. 110–121.
[31] A. Mahmoud and N. Niu, "On the role of semantics in automated requirements tracing," *Requirements Engineering*, vol. 20, no. 3, pp. 281–300, 2015.
[32] P. C. Cozby and S. C. Bates, *Methods in Behavioral Research*, 2012.
[33] P. K. Murukannaiah, N. Ajmeri, and M. P. Singh, "Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in crowd re," in *Proceedings of the International Conference on Requirements Engineering (RE)*, 2016, pp. 176–185.
[34] S. P. F. Fabbri, M. E. Delamaro, J. C. Maldonado, and P. C. Masiero, "Mutation analysis testing for finite state machines," in *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, 1994, pp. 220–229.
[35] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.
[36] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
[37] A. Agresti and M. Kateri, *Categorical data analysis*. Springer, 2011.
[38] Tricentis, "Software fail watch: 5th edition," https://goo.gl/WzXcBe, last accessed: March 2019.
[39] J. Hauke and T. Kossowski, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones geographicae*, vol. 30, no. 2, pp. 87–93, 2011.
[40] J. Cleland-Huang, "Mining domain knowledge [requirements]," *IEEE Software*, vol. 32, no. 3, pp. 16–19, 2015.
[41] T. Bhowmik, N. Niu, A. Mahmoud, and J. Savolainen, "Automated support for combinational creativity in requirements engineering," in *Proceedings of the International Requirements Engineering Conference (RE)*, 2014, pp. 243–252.
[42] T. Bhowmik, N. Niu, J. Savolainen, and A. Mahmoud, "Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering," *Requirements Engineering*, vol. 20, no. 3, pp. 253–280, 2015.
[43] R. K. Yin, *Case study research: Design and methods*. SAGE Publications, 2008, vol. 5.
[44] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, and M. Oivo, "Empirical software engineering experts on the use of students and professionals in experiments," *Empirical Software Engineering*, vol. 23, no. 1, pp. 452–489, 2018.