Students vs. Professionals in Assisted Requirements Tracing: How Could We Train Our Students?

Mr. Tanmay Bhowmik, Mississippi State University

Tanmay Bhowmik is a Ph.D. candidate in the Department of Computer Science and Engineering at Mississippi State university. He obtained his M.S. degree in Computer Science from the same department in 2010. He completed his Bachelor's degree in Computer Science and Engineering from National Institute of Technology, India, in 2007. His research interest is looking at software engineering from a social information foraging (SIF) perspective. Currently he is exploring stakeholders' social interaction and software productivity from an SIF perspective.

Dr. Nan Niu, Mississippi State University

Nan Niu is an Assistant Professor of Computer Science and Engineering at Mississippi State University. He received his Ph.D. in Computer Science in 2009 from the University of Toronto, where he specialized in requirements engineering for software product lines. His current research interests include information seeking in software engineering, requirements engineering, program comprehension, and software engineering education. He is a member of ASEE and a senior member of IEEE.

Dr. Donna Reese, Mississippi State University

Donna S. Reese received her BS from Louisiana Tech University and her MS and PhD degrees from Texas A&M University, all in computer science. She is Professor and Head of Computer Science & Engineering at Mississippi State University where she has been on the faculty since 1988. Donna is a senior member of ACM and IEEE. She is past chair of the Women in Engineering Division of ASEE. Her primary research interests include recruitment and retention of women and underrepresented minorities within computing and engineering.

Students vs. Professionals in Assisted Requirements Tracing: How Could We Train Our Students?

Abstract

Assisted requirements tracing (ART) skills are essential for new college graduates joining the software industry as their initial assignments often involve substantial tracing-related activities. Although studying human analysts in ART is an emerging research trend, how students might behave differently from software professionals is yet to be investigated. In this paper, we compare the performances, processes, and strategies between students and software professionals in carrying out ART tasks for an unfamiliar system. We observe that both students and professionals performing ART activities generally follow a generic four-phase problem solving process: define the problem, develop a plan, implement the plan, and evaluate the solution. We find that students show significant deficiency in the overall problem solving process, whereas many professionals follow unique and effective tracing techniques in defining the problem, and in developing and implementing the plan. We identify the improvement areas and propose a set of learning activities for Software Engineering students to enhance their tracing skills. We implement two learning activities in a Software Engineering course and report our experience. Our study contributes to the improvement of training students in performing ART and other information-intensive tasks in Software Engineering.

1 Introduction

A recent study by Begel and Simon[1] shows that new college graduates joining large software development organizations generally spend their first several months of employment performing corrective and perfective maintenance tasks. Finding the right piece of source code relevant to the change request in an unfamiliar software project is among the initial challenges faced by such new developers. Thus, it is crucial for the Software Engineering educational program to equip the students with core skills to effectively and efficiently locate a concern in the code base and relate the code to other Software Engineering artifacts.

The field of tracking a concern throughout the development life cycle is known as *software traceability*. This line of research has its root in Gotel and Finkelstein's seminal work[10] on *requirements tracing*, where traceability is defined as the "ability to describe and follow the life of a requirement, in both a forwards and backwards direction". The traceability information not only facilitates the identification of the code in response to a change request, but also supports many Software Engineering activities, such as risk assessment, regression test selection, and so on.[11]

As in many Software Engineering activities, automated tools have been developed to support the establishment and management of the traceability information. Although modern tools employ information retrieval (IR) techniques to automatically generate candidate traceability links between software artifacts, the links must be certified by the software engineer to ensure the quality of the traceability information. *Assisted requirements tracing* (ART) thus refers to the process where an engineer engages with the tracing tool to perform the assigned task.[8] In software assurance scenarios, ART is believed to be the only way in which tracing can be at least partially automated.[8]

Recent ART studies[5,8] show that humans do override the tool's output and have varied performances if the initial traceability information is of different qualities. These studies are conducted by observing Software Engineering students who act as software engineers for the subject system. While certain behavioral patterns are uncovered among the students, little is known about how students may perform differently from the professionals. Directly comparing the behavior of students with that of professionals in ART can recognize the strengths exhibited by the students, and more importantly, can reveal areas for improvement. The findings, in turn, will reinforce valuable educational experiences, as well as open new educational avenues for preparing Software Engineering students for their starting jobs in industry.

In this paper, we first report a study that compares the performances, processes, and strategies between students and professionals in carrying out ART tasks. We ask ten Software Engineering students and ten professional software developers to interact with an IR-based tool so as to complete a set of requirements-to-source-code tracing tasks for an unfamiliar system. We not only unobtrusively log the participants' fine-grained interactions to obtain quantitative data, but also employ qualitative methods like observations, protocol analysis, and interviews to gain insights into the participants' tracing behaviors. Based on the findings of this initial study, we incorporate two learning activities into the Introduction to Software Engineering course. We conduct a further study on the students who completed the course in order to assess the effect of our learning activities.

The contributions of our work lie in the direct comparison of students' and professionals' ART behaviors and the concrete recommendations for improving Software Engineering education. We design and integrate specific learning activities into a Software Engineering course and provide first hand evidence of the effectiveness of these learning activities through a followup study. Furthermore, the educational implications drawn from our study have the potential to be related to a wide spectrum of courses in Software Engineering. In what follows, we describe in Section 2 why studying human factors is important for traceability research and why sufficient training in ART is important for new graduates. Section 3 presents our research questions and setup of the initial study. Section 4 discusses the results. Section 5 details the educational implications of our findings in light of pedagogical principles and learning activities. Section 6 describes the implemented learning activities and Section 7 presents the study that explores the effect of these learning activities. Section 8 details the limitations of our overall work, followed by some further discussion in Section 9, and finally, Section 10 concludes the paper.

2 Background and Related Work

In software development, many artifacts can be transformed into a textual format. Establishing plausible traceability links between the textual artifacts can then be casted to an information retrieval (IR) problem, where the artifact to be traced (e.g., a specific requirement) is treated as a query and the IR algorithm searches and returns relevant targeting artifacts (e.g., Java classes that implement the query requirement). Research focused on the underlying IR mechanisms is often called "the study of methods".[11] Extensive empirical studies indicate that all the exploited IR methods so far are equivalent in that they are able to capture almost the same traceability information.[11,15]

As our understanding about the IR-based tracing algorithms evolves, a new area of interest has emerged: "the study of human analysts".[5] In fact, the candidate traceability links generated by an automated tool must be examined and validated by the human. The software engineers, including Software Engineering students who will become future engineers, must vet the automated tool's output and add/remove links as necessary to arrive at the final traceability information. In this sense, the tool only assists the human who plays an active and primary role in making the final decision. Such a process is called *assisted requirements tracing* (ART).[8] Recent studies on students' ART performances[5,8] clearly showed the challenges in that students invariably made errors of omission (threw out correct links) and errors of commission (added incorrect links). Dekhtyar *et al.*[8] conducted a statistical analysis of the factors affecting ART performance, though all the participants were students enrolled in Software Engineering courses. Our work, reported, in this paper extends the body of knowledge in ART by making a head-to-head comparison between students and software professionals.

Preparing graduates for a smooth and successful transition toward their roles in the software industry is a goal for many undergraduate Software Engineering programs. Begel and Simon[1] studied the problems faced by new college graduates in their first software development jobs. Observing eight new developers at Microsoft Corporation showed that new graduates were mostly assigned to corrective, and sometimes perfective, maintenance tasks in their first six months of employment. Similar results are also reported in the literature.[2,6,13,17] Although the new graduates demonstrated good coding abilities, the initial struggles they went through were due to insufficient training and practice for software maintenance assignments. Focusing on maintenance as a whole, Begel and Simon[1] provided several suggestions to improve the students' social and technical skills to mitigate their initial struggles. In this paper, we examine ART at the requirements-to-source-code level, which can be seen as a more focused investigation along the maintenance spectrum. This is because, before attempting any modification, engineers must locate and understand the parts of the software system relevant to the desired change. In many cases, the relevant parts include not only source code but also other artifacts, e.g., requirements.

Incorporating the teaching materials that are of industrial strength is an important education strategy to ensure practical relevance. For example, Way[18] developed a course that implemented a company-based framework in developing a software project. The course objective was to simulate the collaborative framework followed by industry in order to train students to become successful developers in actual software development organizations or research laboratories. Another example was the collaborative course projects proposed by Rusu *et al.*[16] The goal was to provide the students with an opportunity to be familiar with industry settings, and to help them develop their collaboration and team work skills critical to the practice in industry. Chenoweth *et al.*[3] incorporated the idea of multiple role-teams in Software Engineering courses and adapted cooperative learning techniques to improve collaboration skills among students. Georgas[9] implemented active learning and role playing techniques in a course and provided the students with appropriate industrial contexts. A common aspect of industrial practice is the leverage of automated tools, such as debugger and version control support. As in ART, these tools only assist engineer's decision making. Our findings can therefore be applied to a wider range of tasks involving human interaction (reaction) with the automated tool's results. In Section 5, we further elaborate how our proposed pedagogical techniques can be generalized.

3 Research Methodology

The overall objective of our research is to identify improvement areas for the students performing ART tasks and to assess specific learning activities designed to enhance students' ART skills. In the rest of this section, we discuss our initial research questions, as well as the instrumentation and study setup.

3.1 Research Questions

The initial goal of this research is to explore the commonalities and variabilities between Software Engineering students and software professionals in performing ART tasks. Two specific research questions are investigated.

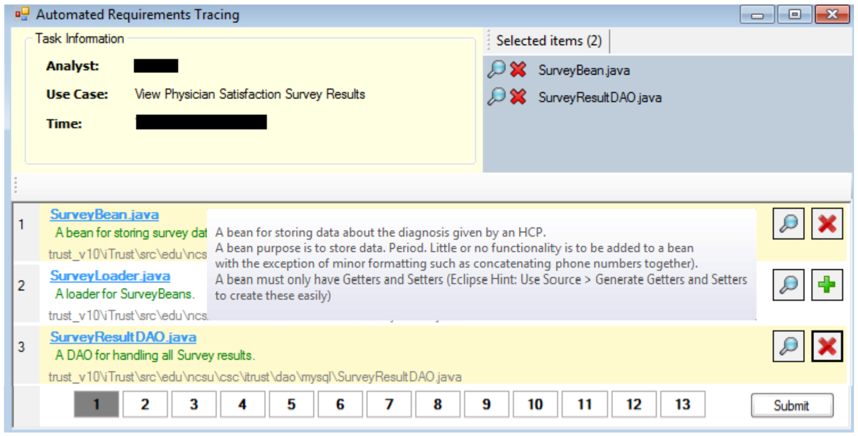
- ***RQ1*:** Is there any significant difference in tracing performance between students and professionals?
- ***RQ2*:** What are the strengths of students and what are the areas that students should improve for developing better ART skill sets?

To answer RQ1, we perform quantitative analysis of the final traceability information submitted by the students and the professionals. The metric used here is the F_2 measure commonly adopted in traceability research.[5,8,11] Formally, let A be the answer set of correct traceability links and B be the set of links submitted by the human. Then, recall is $R = (|A \cap B|)/(|A|)$, and precision is $P = (|A \cap B|)/(|B|)$. F_2 represents a harmonic mean of R and P and is defined as $F_2 = (5 \cdot R \cdot P)/(4 \cdot P + R)$. Note that the F_2 measure weights recall (R) twice as much as precision (P). This is because in automated tracing, it is easier to remove incorrect links than to find missing links.[11]

To answer RQ2, we perform qualitative analysis by collecting a hybrid of data. Our main data sources are observations and notes taken during all the tracing sessions, coding and classifying the fine-grained interactions logged by the tracing tool, analysis of the think-aloud protocols, and exit interviews to elicit participants' opinions and experiences. Note that we employ the commercial Camtasia screen capturing tool to record the step-by-step operations and the think-aloud verbalizations of the participants. A researcher spends about 63 hours in total to keenly watch and encode all the recorded sessions. A second researcher is involved only when clarifications are needed.

3.2 Instrumentation and Study Setup

We developed an automated tracing tool and used it in our study. A screenshot is shown in Figure 1 where the analyst's name and the actual tracing time are not released in order to protect anonymity. We call the tool "ART-Assist" to emphasize the integral yet supportive role it plays in ART. As different IR-based traceability recovery methods show comparable performance,[15] the back-end of ART-Assist adopts the vector space model with the TF-IDF weighting.[11] The front-end uses the ordered list to display the retrieved candidate traceability links according to the similarity score computed by the IR algorithm. Our study examines the traceability between requirements-level use cases (UCs) and implementation-level Java classes.
In ART-Assist, each link's snippet consists of 3 lines: the class name, the class header comment trimmed in 1 line, and the class path which can act as the URL for the retrieved link. A


Automated Requirements Tracing

Task Information

Analyst: [redacted]

Use Case: View Physician Satisfaction Survey Results

Time: [redacted]

Selected items (2)

SurveyBean.java

SurveyResultDAO.java

1 SurveyBean.java
A bean for storing survey dat
trust_v10\iTrust\src\edu\ncs

A bean for storing data about the diagnosis given by an HCP.
A bean purpose is to store data. Period. Little or no functionality is to be added to a bean
with the exception of minor formatting such as concatenating phone numbers together).
A bean must only have Getters and Setters (Eclipse Hint: Use Source > Generate Getters and Setters
to create these easily)

2 SurveyLoader.java
A loader for SurveyBeans.
trust_v10\iTrust\src\edu\ncs

3 SurveyResultDAO.java
A DAO for handling all Survey results.
trust_v10\iTrust\src\edu\ncsu\csc\itrust\dao\mysql\SurveyResultDAO.java

Figure 1: Screenshot of ART-Assist — the automated tracing tool used in our study.

mouse-hover over a link's snippet pops up the tooltip that displays the full class header comments, as shown in Figure 1.

ART-Assist's interaction design philosophy is to fulfill engineer's tracing goal while keeping the operations straightforward, accessible, and responsive. Direct navigation to a certain search result page is enabled by clicking the corresponding page number. The highlighted page number shows which page is currently displayed. Clicking the class name in the snippet or the "magnifying glass" icon allows the entire class file to be viewed in a new window. A link can be selected or deselected via "+" (add) or "×" (remove). A shopping-cart-like area in ART-Assist's upper-right corner enables the explicit management of selected links. Once a link is selected, its snippet is yellow highlighted. When the engineer is satisfied with the selected set of links, the "submit" button shall be pressed.

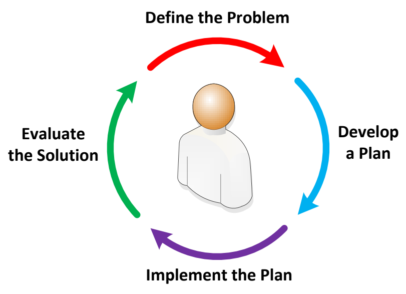
The subject software system of our ART study is iTrust (http://agile.csc.ncsu.edu/iTrust). iTrust is a medium-sized Java application aimed at providing patients with a means to keep up with their medical records. The entire dataset contains 46 UCs and 226 Java classes. The answer set of the requirements-to-source-code traces is prepared by iTrust's developers and has 314 correct links. From the 46 UCs, 6 are randomly selected to serve as the query requirements in our study. Table 1 lists these requirements tracing tasks, together with the number of correct traceability links for each UC as defined in the answer set.

We recruited 10 professional software developers from two local companies. In order to respect our confidentiality agreement, we do not disclose their identities. The professionals had 2 to 5 years of software development experience. Among the recruited professionals, 5 mostly used C# and the rest used Java as their primary development language. Meanwhile, 10 Software Engineering undergraduates, taking the Introduction to Software Engineering course offered at our institute in Spring 2013, participated in our study. Note that these 10 students were randomly selected based on the response to a class-wide invitation toward the end of the Spring 2013 semester. No extra credits were given and the student participation was voluntary. The students, however, were encouraged to apply their Software Engineering knowledge, including the materials learned in the course, to perform tracing. One of the main benefits to the students, as well as the professionals, was to get a first hand experience of a state-of-the-art traceability tool. The study was approved by our institutional review board (IRB).

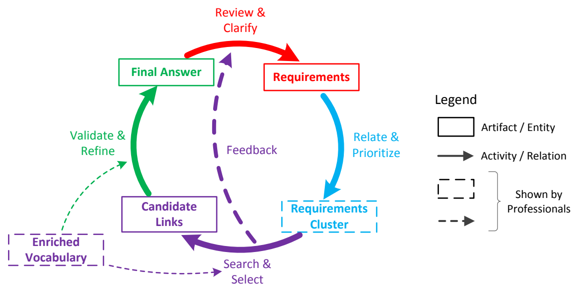
Our institute is a public, comprehensive, and research-oriented university whose Computer

Table 1: iTrust use cases in our study

ID (our study)	Title (iTrust ID)	|correct links|
UC_1	Maintain Standards Lists (UC-15)	13
UC_2	Maintain a Hospital Listing (UC-18)	4
UC_3	View Physician Survey Results (UC-25)	8
UC_4	Document Office Visit (UC-11)	26
UC_5	View Patients (UC-28)	4
UC_6	Safe Drug Prescription (UC-37)	20



(a)



(b)

Figure 2: **(a)** Generic problem solving process. **(b)** Tracing-specific problem solving process.

Science and Engineering Department offers Bachelor's degrees, among others, in both Computer Science and Software Engineering. The Introduction to Software Engineering is a split level course compulsory for Software Engineering majors and involves a lab dedicated to developing a real world Software Engineering project for customers. In total, 25 students took the course in the Spring 2013 semester. None of the participants knew about the iTrust system before the study, but all of them reported being familiar with the healthcare domain. The professionals and the students have had knowledge about traceability and reported a median of 2.5 and 0.5 years of tracing experience respectively.

At the beginning of the study, we made a confidentiality agreement with the participants about not discussing the details of the study with others. During the study, each participant (engineer) worked alone in a lab and began by signing the consent form and by learning how to use the ART-Assist tool. The demographic information was also collected at this stage through a pre-study survey. The information included software development and tracing experience, familiarity with the subject system and the application domain, and the primary and secondary programming languages. The engineer was then given hard copies of the UC descriptions and was told to use only ART-Assist and not to use internet or any other resources over the tracing session. We asked the engineer to trace all 6 UCs and to carry out the tracing tasks in any order they would prefer. A researcher was present to run the ART-Assist tutorial, to encourage the engineer to think aloud during tracing, to take notes, and to conduct an informal exit interview to elicit the engineer's feedback about her tracing experience. Each engineer spent approximately 1 hour completing all the 6 tracing tasks.

4 Results and Analysis

To answer our research questions, we perform both qualitative and quantitative analysis on the collected data. The rest of this section details the results and analysis allowing us to answer these initial research questions.

4.1 Is There a Difference?

As mentioned in Section 3.1, the tracing performance is quantified via the F_2 measure. Our analysis shows that the final traceability links submitted by the professionals have an average F_2 of 0.64 with a standard deviation of 0.05, whereas those submitted by the students have an average F_2 of 0.58 with a standard deviation of 0.06. The result implies a higher quality set of traceability links is determined by the professionals. To assess whether the difference is significant, we follow the work of Dekhtyar *et al.*[8] and use a one-way ANOVA test for the F_2 results between the students and the professionals. The result reveals a statistically significant difference at the 0.05 level ($F = 6.11, p = 0.019$). This indicates that professionals perform significantly better than students in ART. While such a result may not come as a big surprise, the intriguing question is where the differences reside. We answer this question by framing ART in the general problem-solving process.

4.2 Process to Solve Tracing Problems

To gain further insights into the differences, we present our qualitative data analysis results in this section. The answers to RQ2 — strengths of students and areas for improvements — are then presented in Sections 4.3 and 4.4 respectively.

In analyzing the collected data, especially participants' detailed interactions with the ART-Assist tool and their verbalizations during the tracing sessions, we recognize a set of recurring phases of ART. These patterns can be framed in terms of the generic problem solving process shown in Figure 2a. Figure 2b maps the generic process to a more specialized process for ART. The specific phases are described as follows. For the remaining of the paper, we use $\{\mathbf{S_1}, \mathbf{S_2}, \ldots, \mathbf{S_{10}}\}$ and $\{\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_{10}}\}$ to refer to the 10 students and the 10 professionals of this study.

***Define the Problem*:** Every participant started with reading and understanding the requirements described in the UCs. Some reviewed a UC description for a couple of times to understand it clearly, whereas others proceeded to another requirement without spending much time on the current one. For example, just after reading UC$_1$, $\mathbf{S_{10}}$ commented "Don't understand what 'standards' mean in this context" and moved on to UC$_2$. As the participants were free to choose the UCs in any order they would like to trace, three students ($\mathbf{S_2}$, $\mathbf{S_4}$, and $\mathbf{S_9}$) started with a random UC and the other seven took a sequential approach (i.e., proceeded from UC$_1$ to UC$_6$). In case of the professionals, only $\mathbf{P_7}$ began with a random use case, whereas the others went through the titles of all the UCs a couple of times before choosing one to read its detailed description. We refer these tracing activities as *Review and Clarify*.

***Develop a Plan*:** We observed that the students expended little time in or skipped completely the development of a plan for the problem solving. Most students chose a UC, used the ART-Assist tool to generate a list of candidate traceability links, and then inspected the links they thought to be relevant. It turned out that students, in general, quickly dived into the core solution-implementation phase with a very vague notion of developing a plan. On the contrary, professionals showed many identifiable activities in this phase: They reviewed the whole set of the requirements and tried to identify logical relationships among different requirements. A noticeable construct appeared in professionals' problem solving was the *requirements clusters* dividing the UCs into natural groupings. For example, $\mathbf{P_3}$ explicitly stated that he would like to structure his tracing by dealing with three clusters: {UC$_1$, UC$_2$, UC$_4$}, {UC$_3$, UC$_5$}, and

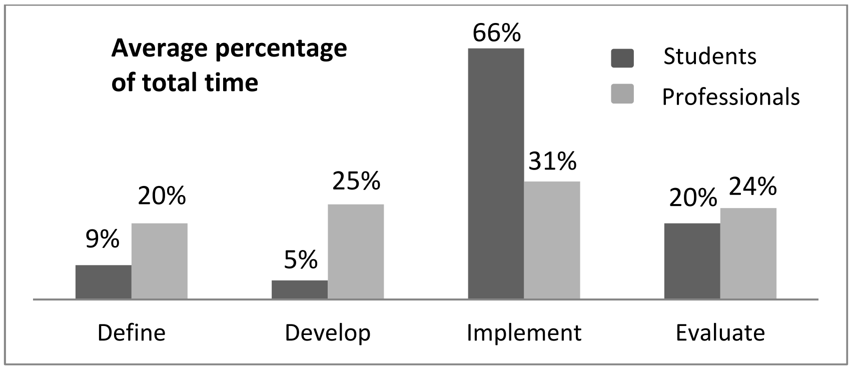


Figure 3: Average percentage of total time spent in different phases.

$\{UC_6\}$. He further commented that he understood the first cluster the best and would therefore tackle that cluster as a whole before attempting to find traces for the other clusters. We name the tracing activities in the plan-development phase as *Relate and Prioritize*.

***Implement the Plan*:** We noticed that students quickly progressed to the solution-implementation phase without much of a concrete plan. They typically spent a significant amount of time in creating the solution. The students mainly browsed the ART-Assist's output and drilled down to specific links that were thought of as relevant. Sometimes they used the tool's search facility to find a keyword or phrase in the Java files. We observed that students usually selected keywords directly from the UC descriptions. Most of the professionals had two main differences in the plan implementation phase. First, as professionals clustered the problems that were related to each other, they tended to solve the problems within a cluster together. This strategy allowed them to reduce the cost imposed by context switches, thereby increasing the efficiency of generating solutions. Second, as the professionals were more aware of the global context, they were able to select their search keywords from an enriched corpus. For instance, when working with UC_3 ("View Physician Survey Results"), searching for "viewPhysician" did not look promising to $\mathbf{P_9}$; changing the keyword to "extractPhysician" did the trick. We use *Search and Select* to refer to the activities involved in the plan implementation and solution generation phase.

***Evaluate the Solution*:** Before submitting their final answers, both the students and the professionals spent some time reviewing the gathered traceability information. Occasionally, the answers were refined before the participants certified their answers. However, many professionals worked differently than students in this phase as well. While the professionals' validation was structured by the requirements clusters, the students reviewed the answers rather randomly. An example was that after modifying the answer to UC_5, $\mathbf{P_3}$ did the same modification for UC_3. In this way, the updates were made in a consistent and complete manner. We call the activities in this phase *Validate and Refine*.

It is worth pointing out that the 4 phases identified above are highly incremental and sometimes interleaved in the tracing sessions. Nevertheless, the pattern of linear progression along the 4 phases is apparent in our data. Figure 3 presents a breakdown of the time spent in each of the phases between the students and the professionals.

4.3 Strengths of Students

Although our results show that the students' overall tracing performance is not as good as the professionals', there are several areas where students demonstrate considerable strength. We next discuss these strong points which the Software Engineering educators shall maintain and reinforce throughout the students' learning experiences.

Persistence. Persistence is among the important strengths commonly shown by the students. This is specially observed in the implementation phase where the students simply do not give up even if they struggle to find the correct answers. They go deep into the list of the candidate traceability links generated by the ART-Assist tool and look into the source code files to obtain useful clues. In Figure 3, the 66% of total time spent in generating solutions indicates the persistent mentality of the students.

Recognition of information sources. Students show the ability of requesting more and varied information sources to help solve the problem. For example, $\mathbf{S_{10}}$ expressed her insufficient understanding about the concept 'standards' in UC_1's description. She further commented that, "I wish I could contact the person who wrote this (UC) . . . I think I can definitely get some help from the internet".

Generation of hypotheses. The students exhibit a strong ability to generate and refine hypotheses during problem solving. If they struggle to find the correct traceability links based on their initial assumptions, they keep modifying and testing the hypotheses, though sometimes the hypothesis-testing cycle lasts for too long. A common hypothesis-refinement strategy observed among the students is to use different terms appeared in the UC description as the keywords for relevance judgement.

Evaluation of answers. Although the students are not found to evaluate multiple answers in a coherent block (cluster), they always dedicate considerable time double checking their solutions before the final submission. Figure 3 shows that, on average, the students spent 20% of the total time in evaluating the answers, a proportion comparable to the 24% that professionals spent in the evaluation phase.

4.4 Improvement Areas for Students

Our study clearly identifies several areas where substantial training and education can lead to enhanced skill set and productivity of students in performing ART tasks. These areas are highlighted by using dotted boxes and arrows in Figure 2b.

1. Clustering requirements in the planning phase. Before going into the traceability link search and selection activities, enough time should be spent reading the requirements, categorizing them based on their similarities, and prioritizing them. Working with a cluster of requirements as a whole instead of individuals one-at-a-time not only reduces the context switch, but also raises the level of abstraction in an effective manner. Such an abstraction plays a crucial role in logically relating the problems and their solutions. In this way, the solutions can be cross-checked efficiently.

2. Shortening the feedback loop. We observe that the software professionals constantly leverage the planning results (i.e., requirements clusters) to avoid wasting too much time in solving difficult or ill-understood problems. In our opinion, this strategy represents the most significant difference between the professionals and the students. Although persistence is valuable, being blindly persistent can be counterproductive. In ART, if the candidate traceability links generated by the automated tool do not look promising, the students should learn to revisit the problem definition or to re-prioritize the requirements (clusters) to be traced. Essentially, this strategy is analogous to engineering design, in which constructing and reasoning about a design model, rather than building the full-fledged system, can help assure quality and minimize re-work. Despite the teaching of the shorten-the-feedback-loop principle in most curriculums, students should learn to apply the idea in a more flexible and dynamic way.

3. Developing an enriched vocabulary. Using an enriched vocabulary to select keywords or phrases for searching relevant information can be of great advantage. Many problems today involve multiple stakeholders who may share their goals, backgrounds, and terminologies only partially. A sample situation is global software development where people with different backgrounds and cultures must work together to achieve project objectives. Being able to develop an enriched, context-aware vocabulary can be a big plus for the students to effectively convey and communicate their core ideas with their colleagues. Thus, learning how to establish semantic relatedness can provide students a competitive edge in solving today's multi-faceted problems, as is certainly the case for ART.

5 Educational Implications

Most Software Engineering courses are designed to teach students the overall software development concepts like the process models and/or specialized methods in architectural design, software testing, etc. However, little is incorporated in the course work to improve the students' skills to trace concerns across a diverse set of software artifacts. In this section, we present a set of educational activities aimed to help students improve not only their tracing skills, but also their skills in several other areas critical to Software Engineering. Table 2 summarizes our proposal based on a combination of pedagogical principles: active learning,[9] collaborative learning,[3] and project-based learning.[7] For each learning activity, Table 2 also corresponds to the improvement areas identified in Section 4.4.

Class discussion for artifact categorization. The learning objective is to raise students' awareness of the interrelationships and interdependencies between software artifacts. For this activity, the focus is on a homogeneous set of artifacts, e.g., a set of requirements-level user stories printed in index cards. This learning activity can be implemented in multiple ways. For example, in one approach, the students can work in small groups to cluster the artifacts, and later on they can share their categorization results to the whole class. In another approach, the whole class can work as a team. The instructor should encourage novel and useful classifications. Meanwhile, if existing (standard) schemes are available (e.g., the checklist of test cases), the students can be actively engaged in learning relevant materials. This activity can be generalized to train students' fault classification and test case selection skills.

Group writing exercises. The learning objective is to help students develop an enriched vocabulary for effective communication. The students are asked to work in small groups to identify the problems and to rewrite some existing (natural language) artifacts. For example, certain help pages of popular products, such as Microsoft Word, could be rewritten to better support the users. In this exercise, the students are encouraged to take multiple stakeholder roles into account. The writing activities can be structured as in-class exercises, homework assign-

Table 2: Learning activities for improving Software Engineering education (the improved areas are referred to the points listed in Section 4.4)

Pedagogy	**Learning activity**	**Area(s) improved**	**Also applicable to**
Active and collaborative learning	Class discussion for artifact categorization	***1***	• Fault classification • Test case selection
	Group writing exercises	***3***	• Bug report • User manual
Project-based learning	Ripple effect reasoning	***2***	• Change impact analysis • Risk assessment
	Conflict resolution	***2, 3***	• Requirements negotiation • Software merging

ments, or both. We envision such (re-)writing exercises can also be targeted to bug reports and user manuals.

Ripple effect reasoning. The learning objective is to enable the students' abilities to make abstractions and shorten the feedback loop by reasoning at a high abstraction level. The students are positioned in a project environment and are asked to make such broad decisions as resource allocation. The essence of this activity is to train the students with making decisions under constraints, thereby explicating the value of abstracting out marginally relevant details and obtaining quick feedbacks. We believe such an activity can be applied to risk assessment, as well as change impact analysis for software projects with managed repositories.

Conflict resolution. The learning objective is to improve the students' synthesized communication and abstract thinking skills. Students are introduced to a multi-stakeholder project that exhibits the nature of concurrent and distributed development. Some long-lived, large-scale, and open-source projects (e.g., Mozilla) could be considered as candidate subjects. Students are exposed to the diverse interests and oftentimes inconsistent views of different stakeholders. Then they are instructed to analyze and eventually resolve the (subset of) conflicts. This activity effectively tackles two areas identified in Section 4.4: shortening the feedback loop and developing an enriched vocabulary. We argue that these improved skills can increase the confidence for students to carry out other important activities such as requirements negotiation and software merging.

6 Implementing Learning Activities

In order to test the effectiveness of the learning activities proposed in Section 5, we implemented two learning activities in the Introduction to Software Engineering course offered in Fall 2013 by the Department of Computer Science and Engineering at our institute. In particular, we incorporated class discussion for artifact categorization and conflict resolution along with the traditional syllabus for the course. We chose these two learning activities as they are expected to cover all three improvement areas for the students (cf. Table 2). We design the learning activities around the Software Engineering project associated with the course (cf. Section 3.2). Implementation of this modified course work provided us an opportunity of assessing the learning activities from a practical perspective. The rest of this section details the

Software Engineering project and the learning activities we integrated in the class as well as in the lab sessions.

6.1 Transportation Security and Resilience Project

This project was developed to aid a group of operation research and optimization researchers in the Department of Transportation and Systems Engineering at our institute. The project was a Google map based system providing tracking and routing facilities for inventory delivery vehicles. It was a semester long project where the whole class worked as a software development team supervised by a teaching assistant (TA) who conducted the lab sessions. In total 24 students took the course. The students were divided into five lab groups and each group was responsible for developing an individual module. The objective of the course project was to create opportunities for the students to obtain hands-on experience on Software Engineering activities. The customers were introduced to the developers (the students in this case) at an early stage of the semester and the lab groups had regular project meetings with the customers throughout the semester. The meetings served various purposes, such as requirements elicitation, negotiation, prioritization, communicating project updates and mutual expectations among the developers and customers, etc. Some of the initial requirements for the system are as follows. We identify them as F_1 through F_6 for further reference.

- F_1**:** Display all the stops on Google map and show shortest path between any two given stops.
- F_2**:** Detect road block or traffic jam ahead on the route with the help of FM traffic alert service.
- F_3**:** Calculate and show shortest rerouted path to the destination in case of adverse traffic conditions.
- F_4**:** Predict the location of a particular vehicle at a given time.
- F_5**:** Detect weakening network signal and dynamically reestablish connection with the strongest available node.
- F_6**:** Provide a facility for the vehicle driver so that he can upload/update his current scenario in the system.

In what follows, we discuss the pedagogical activities: class discussion and conflict resolution, that were meshed with traditional Introduction to Software Engineering course topics.

6.2 Class Discussion for Artifact Categorization

We incorporated this learning activity when requirements engineering (RE) oriented topics, such as requirements prioritization and negotiation were discussed in the class. We designed this learning activity following the pedagogical philosophy that effective learning involves discussion among students, practical work, and practice of important techniques to solve problems.[4,14] In the class, the students were exposed to the overall objectives and available theoretical techniques for requirements negotiation and prioritization. Almost all the requirements from the users were in the form of user stories. By the time of this class discussion activity, the students already had a couple of meetings with the clients and collected a set of user stories for the entire system. The objective of this exercise was to let students discuss every user story, extract specific requirements from the stories, categorize the requirements based on their functional relatedness, and identify the top priority requirements. The class discussion took place for an hour. In order to promote autonomy and professionalism among students, the instructor did not participate actively in the discussion activities but was present to monitor the class.

After the hour long discussion, the students classified the requirements into three categories: networking related requirements (e.g., F_5), shortest path and routing related requirements (e.g., F_1, F_3, and F_4), and current traffic information related requirements (e.g., F_2 and F_6). Most intense discussion (could also be termed as debate in this scenario), took place during requirements prioritization. The instructor observed that there were disagreements among the students about the priorities and implementation difficulties about a couple of requirements. For example, while discussing F_6, some students considered the phrase 'current scenario' to be vague and voted for further discussion with the customers. In contrast, some students argued that this requirement should not be a priority as F_2 should 'arguably' serve the purpose of this requirement. After such debate, the students decided to put aside the requirement for the upcoming customer-developer meeting. Another requirement, "live detection and update of the latest stoppage plan and rerouting accordingly", was considered to be less important by some students and they recommended to drop it from the initial release.

6.3 Conflict Resolution

In a practical Software Engineering scenario, customers tend to use inconsistent and conflicting terms/phrases in order to communicate their needs to the developers. This situation was indeed common in the transportation security and resilience project. For example, the phrase 'current scenario' in F_6 could either mean a sudden change in the traffic condition around the vehicle not yet detected by the FM traffic alert service, or a mechanical problem with the vehicle itself. Students need to be trained to identify and resolve such conflicts in an independent manner to be successful not only during their initial Software Engineering careers but also throughout their professional lives. To that end, we designed this learning activity as an incremental, semester-long practice assignment.

After clarifying different aspects of the application domain through a few initial meetings with the clients, the students were given a lab assignment to create domain dictionaries[12] for the system. The students were divided into five groups, every group was assigned a module of the system and each group came up with their own domain dictionary for the module they were responsible for. The groups were asked to consider every possible term/phrase they could imagine related to the scenarios and to include them in the domain dictionaries. The groups refined their domain dictionaries every week and submitted the latest version at the end of the week to the lab TA. We noticed that more rigorous revisions of the dictionaries occurred after a client-developer meeting and during the overall RE process. Thereby, through this pedagogical activity, the students developed an enriched word bank for the project domain. Such an iterative and evolving process of developing vocabulary was targeted to facilitate critical thinking about the problem scenarios.

7 How Did The Learning Activities Affect Students' ART Performances?

Towards the end of the semester, we conducted another study similar to the one described in Section 3. As our objective was to examine how our designed learning activities influenced ART performances of the students, we limited this study only on the students taking the Introduction to Software Engineering course offered in Fall 2013. However, we did not consider two students for this study who could not make through the course in the Spring semester and had to retake it in the Fall. By not considering these two students, learning effect was better controlled. From the remaining of the class, 10 students were randomly selected to participate in our ART study. The selection process was the same as our Spring 2013 study. Since the only two students who took the same course twice were excluded, there was no overlap between the student participants in Spring 2013 and those of Fall 2013. Similar to the Spring 2013 study, the students reported a median of 0.5 years of tracing experience. All the participants were more or less familiar with the healthcare domain but none of them knew about iTrust or our ART-Assist tool. We used the same set of UCs listed in Table 1 and the same ART-Assist tool to generate candidate traceability links for the UCs.

As in the first study, each participant signed the consent form, learned the ART-Assist tool and worked individually in the lab with their tracing tasks. The students were asked to use only the ART-Assist tool to trace all 6 UCs. Similar to the initial study, a researcher was present to run the tutorial, to make sure the participants think aloud during tracing, to take important notes, and to conduct the exit interview. While the ultimate objective of our first study was to identify the improvement areas for the students, this study was focused on exploring any improvement on the students' ART techniques. The researcher paid special attention to this aspect while taking notes and conducting the exit interviews. Section 9 provides further discussion on these interviews.

Results and analysis: For each participant, we calculate F_2 in order to quantify their tracing performances (cf. Section 3.1). To our surprise, the analysis reveals that the final traceability links submitted by the students in this study (Average $F_2 = 0.63$, $Sd = 0.05$) are almost as good as those of the professionals in the first study (Average $F_2 = 0.64$, $Sd = 0.05$). In fact, two students (F_2 is 0.74 and 0.72 respectively) performed even better than any professional. The average F_2 indicates that higher quality set of traceability links is determined by the participants in the second study than that of the students in the first study (Average $F_2 = 0.58$, $Sd = 0.06$). This overall improvement in the students' ART performances is a manifestation that the learning activities discussed in Section 6 possess a positive influence on training students with better ART skills.

We want to examine if there exists any significant difference between the students' performances in our two studies. The idea being a significant difference between the two groups will testify that the incorporated pedagogical activities help students improve their ART skills in a significant manner. To that end, we perform a one-way ANOVA test[8] between the F_2 results associated with the students from the two studies. The result affirms a statistically significant difference at the 0.05 level ($F = 5.89, p = 0.027$). Therefore, we conclude that the class discussion and conflict resolution activities we included in the Introduction to Software Engineering course significantly help students develop their skills in performing requirements tracing related activities.

8 Threats to Validity

Several factors can affect the validity of our overall exploratory study: construct validity, internal validity, external validity, and reliability.[19] Construct validity concerns establishing correct operational measures for the concepts being studied.[19] The main construct in our case studies are the ART performances of the students and the professionals quantified by the F_2 measure. As the F_2 measure is commonly accepted in traceability research[5,8,11] and as it maintains a proper balance between precision and recall, we expect very limited construct validity issues in our overall study.

Regarding internal validity,[19] our major sources of data were the final traceability links submitted by the students and professionals using our ART-Assist tool, notes taken by the researcher during the study, and the exit interviews. The tool was tested rigorously and we are confident that it kept proper track of the participants' tracing activities. Our reliance on notes and interviews meant that we needed to trust the researcher's observation and each participant's description of her own experience. Participants may have omitted important facts, or we may have misinterpreted them. In order to address this threat, we cross verified the notes with the exit interviews and the recorded audios of the participants' 'think aloud' verbalizations. In an attempt to eliminate learning effect, our second study was conducted on a separate group of students. The tracing experience of these students were similar to that of the students in the first study (median = 0.5 years). However, our overall study does not capture any potential bias due to the students' software development experience. Nevertheless, the Software Engineering background and experience were found not to make a difference in ART.[8] Following this line of reasoning, we did not treat the software engineer's competence and productivity as independent variables in our study. Rather, we made the tradeoff by focusing on the comparison of professionals and students in ART. As far as the two groups of 10 students are concerned, our post-analysis showed that their Software Engineering skills were comparable by the measure of the grades they obtained in respective Introduction to Software Engineering courses (Spring 2013 grade: $mean = 88.5$, $median = 88$, $sd = 5.1$; Fall 2013 grade: $mean = 82.5$, $median = 86.5$, $sd = 9$).

Another important factor that needs to be considered is the sample size, i.e., 10 students and 10 professionals in each of our study. Contemporary empirical research in ART had 38[8] and 26[5] student participants. While our studies had comparable numbers, having 10 professionals carrying out ART tasks was unique in our settings. Having more participants definitely increases the power of statistical analysis, but we argue that the quantitative differences (in terms of the F_2 measures) and the qualitative ones uncovered by our work made a necessary and key initial step for understanding ART performed by different groups.

External validity concerns establishing the domain to which a study's findings can be generalized.[19] There should be difference in the quantitative results presented in Section 4 and Section 7 in case of studies with different sets of participants and software systems. However, the knowledge we have discovered should be similar in other studies performed in similar settings. Although there might be difference in the level of improvement of the students' tracing skills, we expect that our learning activities will hold similar effects on new Software Engineering student bodies.

Reliability of a study suggests that the operations of a study can be repeated with the same results.[19] Due to the nature of our study, an exact replication may not be possible in different settings. However, the underlying trends and implications presented in this paper should re-

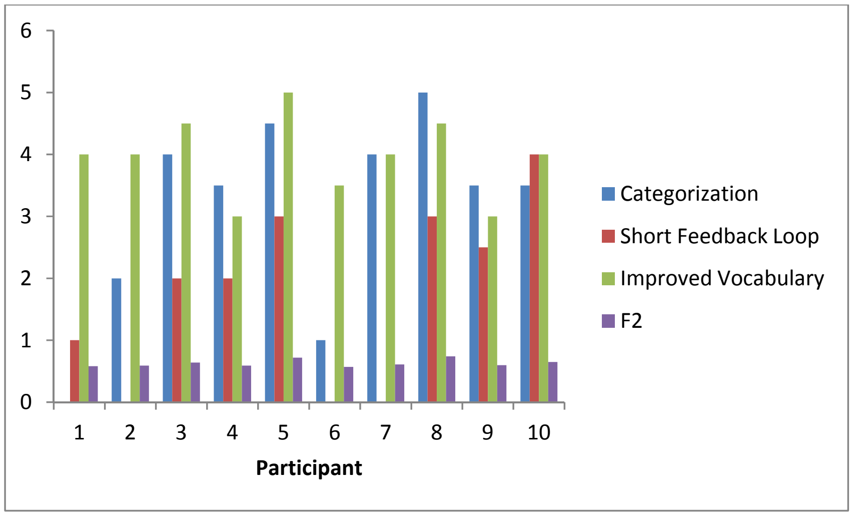


Figure 4: Average ratings associated with each participant along with the corresponding F_2.

main unchanged, with an exception that different choice of requirements (e.g., selecting a set of completely unrelated UCs of iTrust) may not benefit equally from clustering.

9 Discussion

The results and analysis presented in our study indicate that many professionals follow improved tracing techniques and outperform untrained students in conducting ART activities. Our initial study also identifies three areas: clustering requirements in the planning phase, shortening the feedback loop, and developing an enriched vocabulary; where students could be trained to enhance their ART skill sets and productivity. The researcher conducting the second study asked the students explicit questions during the exit interviews regarding the improvement areas. In the rest of this section, we discuss how the learning activities affected the three improvement areas for the students. We use $\{\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_{10}}\}$ to refer to the participants in our second study.

***Clustering requirements in the planning phase*:** During the class discussion for the artifact categorization exercise, the instructor did not provide any clue that there might be any study on this skill. However, we observed that many of our participants indeed clustered the UCs at an early stage of their tracing activities. In fact, $\mathbf{s_8}$ ($F_2 = 0.74$) explicitly mentioned "probably it is a good time for me to apply something I learned in the class". During the exit interview, every student was asked the question, "Did you apply some sort of categorization on the UCs during your tracing activities? If yes, please rate the extent of your categorization at a scale of 0 to 5 with 0 being no categorization and 5 being the use of explicit categorization". Although a few participants indicated 0 or very low ratings (e.g., both $\mathbf{s_1}$ and $\mathbf{s_6}$ rated 0), most of the better performers were found to be using some sort of clusters of the UCs. It should be noted that a student could use a tracing technique without being explicitly aware of her action (e.g., $\mathbf{s_6}$ unknowingly used clustering to some extent). As this could have a bias on the student's rating, the researcher conducting the study also provided a rating independent of the participant's opinion (e.g., in case of $\mathbf{s_6}$, the researcher's rating for clustering was 2). We calculate the average of these ratings and present them in Figure 4 for each participant. Figure 4 shows that higher F_2 has relatively higher rating for categorization.

***Shortening the feedback loop*:** It should be noted that the learning activity we integrated into the course work was not specifically targeted to improve this tracing skill. However, based on our observation and understanding of ART activities, we expected that the learning activities may influence this particular tracing skill as well. A few participants appeared to be using this technique. For example, while working with UC_6 (total number of correct links is 20), $\mathbf{s_5}$ commented “looks like a complicated one... should not spend more time on this” and moved on to UC_3. Nevertheless, we observed that the use of this tracing technique was not up to the mark among the participants. Our exit interview question relevant to this area was “Did you find some UCs particularly difficult than others? If yes, did you save some time working less on them and utilized that time on more promising UCs? If yes, please rate the extent to which you applied this strategy at a scale of 0 to 5.” Four participants sounded confused with this question. In fact, we did not obtain any rating for 3 participant (cf. Figure 4). In our opinion, these findings suggest two points: 1) the students were not explicitly cautious about the idea of shortening the feedback loop, and 2) learning activities specifically designed to improve this tracing technique among students is warranted.

***Developing an enriched vocabulary*:** Compared to our first study, we found noticeable improvement in building and using an improved vocabulary among the students during the second study. It appeared that almost all the students considered synonyms and alternative key words while performing the tracing tasks. Considering the vocabulary aspect, we observed some striking similarities among the professionals in the first study and the students in the second study. For instance, while working with UC_3, $\mathbf{s_8}$ also considered the keyword “extractPhysician” as $\mathbf{P_9}$ did during the initial study. The exit interview question corresponding to enriched vocabulary was “To what extent did you use various terminologies based on your domain knowledge? At a scale of 0 to 5, please rate your extent of using this enriched vocabulary.” The researcher also provided his individual ratings. Figure 4 suggests that the use of a rich vocabulary among students was indeed dominating.

10 Conclusion

In this paper, we present a study that compares students’ and professionals’ ART behaviors. Based on our initial experiment, software professionals perform significantly better than the students in completing the requirements-to-source-code tracing tasks for an unfamiliar system. We identify the strengths of students, and also find that there are several unique strategies many professionals adopt, such as clustering requirements in the planning phase, shortening the feedback loop, and using an enriched vocabulary. Equipping the Software Engineering students with these strategies can increase their performances and productivities. We further propose pedagogical principles and specific learning activities to show how the strategies and the skill sets can be taught in a Software Engineering context.

We implement two of our proposed learning activities: class discussion for artifact categorization, and conflict resolution in the Introduction to Software Engineering course offered in a subsequent semester at our institute. We conduct a further experiment in order to assess the effect of these pedagogical activities on the students’ tracing skills. The results suggest that our proposed activities indeed help students improve their performance in conducting ART related tasks. In particular, we observe significant improvement in the students’ requirements-to-source-code tracing performances by improving their use case categorization abilities and by enriching their vocabulary skills. However, our study does not capture substantial evidence that supports development in “shortening the feedback loop” skill among students. We plan to conduct further studies regarding this tracing technique. In other words, we want to design and implement learning activities specifically aiming at training students with this tracing skill. We also plan to assess the effect of those learning activities in our upcoming study. Finally, we are also interested to investigate the long-term effectiveness of the skills instilled by our learning activities.

References

[1] Begel, A., and Simon, B. Struggles of new college graduates in their first software development job. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education (SIGCSE)* (March 2008), Portland, Oregon, USA, pp. 226–230.

[2] Beus-Dukic, L. Final year project: a test case for requirements engineering skills. In *6th International Workshop on Requirements Engineering Education and Training (REET)* (2011), pp. 5–8.

[3] Chenoweth, S., Ardis, M., and Dugas, C. Adapting cooperative learning to teach software architecture in multiple-role teams. In *Proceedings of ASEE Annual Conferences & Expositions* (2007).

[4] Committee on the Mathematical Sciences in the Year 2000, N. R. C. *Moving Beyond Myths: Revitalizing Undergraduate Mathematics*. The National Academy Press, 1991.

[5] Cuddeback, D., Dekhtyar, A., and Hayes, J. H. Automated requirements traceability: the study of human analysts. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)* (September 2010), Sydney, Australia, pp. 231–240.

[6] Dagenais, B., Ossher, H., Bellamy, R. K. E., Robillard, M. P., and Vries, J. P. D. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (2010), ACM, pp. 275–284.

[7] DeFranco, J. F., and Neill, C. J. Improving individual learning in software engineering team projects. In *Proceedings of ASEE Annual Conferences & Expositions* (2013).

[8] Dekhtyar, A., Dekhtyar, O., Holden, J., Hayes, J. H., Cuddeback, D., and Kong, W.-K. On human analyst performance in assisted requirements tracing: statistical analysis. In *Proceedings of the 19th IEEE International Requirements Engineering Conference (RE)* (September 2011), Trento, Italy, pp. 111–120.

[9] Georgas, J. Supporting software architectural style education using active learning and role-playing. In *Proceedings of ASEE Annual Conferences & Expositions* (2013).

[10] Gotel, O., and Finkelstein, A. An analysis of the requirements traceability problem. In *Proceedings of the 1st IEEE International Conference on Requirements Engineering (RE)* (April 1994), Colorado Springs, Colorado, USA, pp. 94–101.

[11] Hayes, J. H., Dekhtyar, A., and Sundaram, S. K. Advancing candidate link generation for requirements tracing: the study of methods. *IEEE Transactions on Software Engineering 32*, 1 (January 2006), 4–19.

[12] Lee, K., Kang, K. C., and Lee, J. Concepts and guidelines of feature modeling for product line software engineering. In *Software Reuse: Methods, Techniques, and Tools*. Springer, 2002, pp. 62–77.

[13] Merten, T., Schafer, T., and Bursner, S. Using RE knowledge to assist automatically during requirement specification. In *7th International Workshop on Requirements Engineering Education and Training (REET)* (2012), pp. 9–13.

[14] Moore, D. S. New pedagogy and new content: The case of statistics. *International statistical review 65*, 2 (1997), 123–137.

[15] Oliveto, R., Gethers, M., Poshyvanyk, D., and Lucia, A. D. On the equivalence of information retrieval methods for automated traceaibility link recovery. In *Proceedings of the 18th IEEE International Conference on Program Comprehension (ICPC)* (June 2010), Braga, Portugal, pp. 68–71.

[16] Rusu, A., Rusu, A., Docimo, R., Santiago, C., and Paglione, M. Academia-academia-industry collaborations on software engineering projects using local-remote teams. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE)* (March 2009), Chattanooga, Tennessee, USA, pp. 301–305.

[17] Sikkel, K., and Daneva, M. Getting the client into the loop in information system modelling courses. In *6th International Workshop on Requirements Engineering Education and Training (REET)* (2011), pp. 1–4.

[18] Way, T. P. A company-based framework for a software engineering course. In *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE)* (February 2005), St. Louis, Missouri, USA, pp. 132–136.

[19] Yin, R. K. *Case study research: Design and methods*, vol. 5. SAGE Publications, 2008.