Strengthening Elements of Teamwork, Innovation, and Creativity in a Software Engineering Program

Kevin Daimi*

Abstract - Current software engineering concepts, techniques, processes, and methods are the outcomes of various kinds of innovations, creativity and teamwork. This paper attempts to promote teamwork, innovation and creativity practices among students. The paper also aims to provide a model that faculty can incorporate in their software engineering courses. The innovational approaches of painstorming, critical thinking, case methods, problem-based learning, trimming techniques, bisociation, and opportunity recognition will be introduced. Each approach will be supplemented by some examples from the software engineering domain. These approaches and their accompanying examples aim to develop a number of entrepreneurial-mindset attributes.

1. Introduction

Software Engineering is the field dealing with constructing and maintaining reliable, secure and quality-based software systems. These systems should be cost-effective to build and sustain. It is vital that such systems fulfill all the customer’s needs to be considered as valuable and effective solutions for customer’s problems (Pfleeger and Atlee, 2010; Sommerville, 2011). The field of software engineering has become extremely complex, integrating many elements including technical features, management skills, and stakeholders’ concerns (Bernhart, et al., 2006).

When designing software engineering curriculum, the Software Engineering Body of Knowledge (SWEBOK) recommends that ten knowledge areas be included in the curriculum: software requirements, design, construction, testing, maintenance, configuration management, engineering management, engineering processes, engineering tools and methods, and quality (Guide to Software Engineering Body of Knowledge, IEEE, 2005). Further recommendations for designing undergraduate software engineering curriculum can be found in the 2004 Software Engineering Curriculum Guide (Foster and Lin, 2003). Unfortunately, neither of these curriculum guidelines places any emphasis on the value of innovation in software engineering.

Bridging the gap between software engineering education and industry is an indispensable curriculum design principle (Honiden, et al., 2007). To achieve that, collaboration between industry and academia throughout all the design stages is essential. Software engineering practitioners have an enormous role to play in advancing the future and the practical approach of software engineering education (Lethbridge, et al., 2007). An industry vision on software engineering education is provided by Reifer (2005). The author’s viewpoints concentrate on the need for software engineers who are problem solvers, possess excellent communication skills, and work collectively with users in an interdisciplinary environment; these are characteristics that frequently arise when describing entrepreneurially minded engineers. Although these studies value the interactions between software engineering departments and industry, they ignore the fact that academia can definitely introduce students to innovation and entrepreneurship and begin to cultivate the entrepreneurial mindset in students.

* Professor and Director - Software Engineering Programs
University of Detroit Mercy
Entrepreneurship is exemplified in a skill set that is contributing to recognizing opportunities and creating successful ventures to make the most of such opportunities. Opportunities are prerequisites to successful entrepreneurship ventures, and the ability to identify and assess opportunities is a prerequisite skill to an entrepreneur. The importance of opportunity is discussed amply in the literature (Gartner, et al., 2003; Dutta, and Crossan, 2003). From an entrepreneurship education standpoint, educators can implant the basics of entrepreneurship by training the students to recognize opportunities (and ultimately providing them with the knowledge and tools to act upon those opportunities).

Innovation is inspired by the ability to observe connections, to identify opportunities, and to investigate possible ways to act on these opportunities. It involves three principal premises: creating new ideas that address the opportunities, screening to identify the most promising idea(s), and implementing them (Bassant and Tidd, 2007). According to Carlson and Wilmot (2006), “Innovation is the process of creating and delivering new customer value in the marketplace.” We need to promote innovation throughout the Software Engineering program. This will be accomplished through implanting the foundational practices of innovation into our Software Engineering curriculum so that students can learn innovation by doing. Business leaders look upon innovation as a core competency, and the only way to guarantee marketplace strength for lean organizations (Denning, 2004).

To incorporate innovation, creativity, and entrepreneurship in the classroom, researchers have suggested a number of approaches including designing courses or developing term projects. Rohde, et al., (2005), and Klamma, et al., (2003) focused on launching communities of practice among students and start-up companies. To this end, they developed the course “High Tech Entrepreneurship and New Media” in an effort to sustain the practice of social-capital building between entrepreneurs and students. They offered projects based on the experience of local start-up companies and concluded that having start-up entrepreneurs involved in the group projects would simulate the market-oriented perception within the course.

Foster and Lin (2003) introduced a project, “Managing Innovation in the Digital Economy,” to augment the undergraduate and graduate information management and information systems curricula with entrepreneurial ideas. They inferred that “differences in levels of prior knowledge of business studies and in cultural background can impact on students’ acquisition of domain knowledge and intellectual and information research skills during collaborative development of a business plan.”

The vast majority of software engineering programs do not offer entrepreneurship or innovation courses. Rusu, et al., (2006) proposed a framework to give software engineering students the opportunity of experiencing and igniting the spirit of entrepreneurship early in the program. Their outline included the following phases: spotting potential enterprising individuals, assigning students to entrepreneurial mentors, demonstrating entrepreneurial skills within a software engineering course, and having students develop their own entrepreneurial ideas.

Kussmaul (2000) introduced a team project within their Systems Design and Implementation course to stress software entrepreneurship. In addition to the main goal of working in substantial significant software projects and exercising software engineering and project management, students are also be exposed to ideas and concepts of entrepreneurship within the Systems Design and Implementation course. This paper endeavors to enrich our software engineering education with innovational thinking, creativity, and teamwork. As in any other engineering field, innovation and creativity play a major role in shaping our software engineering techniques, processes, procedures, methods, and concepts. The following innovation approaches will be covered: painstorming, critical thinking, case methods, problem-based learning, trimming techniques, bisociation, and opportunity recognition. Each approach will be accompanied by some examples. The examples will directly address what is introduced in the software
engineering textbooks in order to enhance them with innovational thinking. For all the examples presented below, the instructor briefly explains to the class that the improved or new product is driven by market needs/expectations and opportunities. The problem sheet for each example states that “Current market need/expectation/opportunity has evolved to demand product improvement or creating new product. Develop concepts to transform the existing solution to address these needs.” Furthermore, the presented examples directly relate to the following entrepreneurial-mindset (e-mindset) student outcomes: effectively collaborate in a team setting, apply critical and creative thinking to ambiguous problems, and persist through and learn from failure. This work has relied on a number of software engineering textbooks: Laplante (2009); Pfleeger and Atlee (2010); Sommerville, (2011); and Taylor, et al., (2010). In addition to developing innovational thinking, creativity, and teamwork among students, faculty can use the presented examples as templates to create their own examples and exercises.

2. Painstorming

Painstorming refers to focused brainstorming where a pain in the marketplace is identified as a potential opportunity for innovation. It involves the detection of daily hardship that might be mitigated by the new product or service (Ryckman, 2010). Painstorming could also be thought of as a technique for identifying problems (Benson, 2010). Based on these views, it is concluded that painstorming is a method for establishing customer’s needs. Painstorming can help an engineer “avoid answering a question nobody asked.” In other words, focusing on a true pain/opportunity in the market increases the likelihood of developing a high-value solution. The following examples should illustrate the role painstorming can play in software engineering education.

2.1. Example 1

Figure 1 represents a design of a burglar alarm system. It consists of six components: movement sensors, door sensors, alarm controller, siren, voice synthesizer, and telephone caller (fixed line). Students are asked to specify the “pains” (hardships or problems). Two pains are obvious: there is no battery backup, and there is no cell backup. These two problems are critical in case the burglar succeeds in disconnecting the fixed line phone, or there is a power failure.

![Figure 1. Burglar Alarm System](image)

Based on the identified pain, the Burglar Alarm System is improved to meet customers’ needs as in Figure 2.
2.2. Example 2

In this example, students are divided into three groups: software maintainers, software testers, and software designers. The software designers display the sales transaction design (Figure 3) below. Maintainers and testers are asked to indicate the pain.

The software maintenance and software testing groups should determine that for the `generateBill` method to print the list of items sold, it should access and be directly dependent on the Sale, Customer Account, and Item classes. This dependency will demand knowing the interface of each of these classes in order to invoke the needed methods, and re-implementing the `generateBill` method should any of these classes be modified. As a result, the software designer group should improve their design by introducing two new methods; `printSalesItems` and `printItemList` as illustrated in Figure 4.
3. Informal Cooperative Learning

Informal cooperative learning is used to motivate student attention, assist in establishing expectations, guarantee cognitive processing, and grant conclusion to instruction (Johnson, and Johnson, 1996). Klein and Schnackenberg (2000) investigated the outcome of informal cooperative learning and the association motive on accomplishment, reaction, and student communications. Informal collaborative learning can encourage creative and critical thinking. It can also set the foundation for generating innovative ideas. According to Slavin (1996), “Students will learn from one another because in their discussions of the content, cognitive conflicts will arise, inadequate reasoning will be exposed, and disequilibration will occur.” Examples of informal collaborative learning in software engineering that include critical thinking are presented below.

3.1. Example 1

Using the Burglar Alarm System in Figure 1, student groups are asked to:

a. Estimate how much computer memory is needed for the Alarm Controller component.

b. Specify the time taken by each control function to execute.

Certainly, we are not interested in the figures themselves, but in how these figures are reached—both in terms of technical logic and team dynamics. This example also involves critical thinking. Groups should estimate the number of instructions based on the control functions. Students then need to know how many bytes each instruction takes, and how long each instruction takes when executed. The instructor might observe and subsequently discuss various interesting aspects of the group dynamics while working on their estimates.

3.2. Example 2

In Figure 5, components of an Insulin Pump System are numbered. The data items are provided in Table 1. The cells of the “Type” columns should be filled with the component number followed by either I (input), O (output) or IO (input/output). For example, 1O means it is an output of component 1, 4I means an input to component 4, and 4O3I implies an output of component 4 and an input to component 3.
First, individual students work on filling out the cells. Then students work in pairs. Finally, it is a team effort with teams formed by joining four people per team—no two of whom have been previously paired (so each team brings the collective knowledge of eight students). The instructor provides the right answers and the percentages of answers in the columns with the heading “d.” Typically, the data shows increased accuracy of the pairs over the individuals and of the teams over the pairs. This helps build an understanding of the value and process of collaboration.

### Table 1. Data Items for Insulin Pump

<table>
<thead>
<tr>
<th>Data item</th>
<th>Individual</th>
<th>Pair</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump Control Commands</td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insulin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood Parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood Sugar Level</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insulin Requirement</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.3. Example 3

Table 2 contains the steps of the process for capturing requirements. Rank these from 1-5, where “1” represents the first step in the sequence and “5” the last one. Again, the exercise is solved by individual students, pairs, and teams.

### Table 2. Steps for Capturing Requirements

<table>
<thead>
<tr>
<th>Step</th>
<th>Individual</th>
<th>Pair</th>
<th>Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elicitation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Requirements Specification (SRS)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Case Methods

In software engineering, cases are narratives that provide information about software development history, techniques, methods, success, and failure. Cases exclude any analysis or conclusions. Butler (1999) developed a case study to improve students’ analytical capabilities in software design through focusing on software engineering problems. Cases can permit students to develop excellent reasoning skills, deal with real-world examples in the classroom, provide the opportunity for students to learn by doing, and result in organizational impacts, social values, and ethical issues to be brought to the forefront of discussion (Hackney, McMaster, and Harris, 2003).

There are many case studies available on the Internet and in software engineering books. However, care must be taken when adopting these case studies. The case method requires the case description to exclude any conclusions and analysis. This is the main difference between a case and a case study. Therefore, case studies need to be checked for these conditions. If a case study involves analysis and conclusion, the case study must be re-written to remove the analysis and conclusions. Cases can also be created from papers, lecture notes, and the faculty member’s experience. Another approach would be to consult the software industry. A number of software engineering cases and their design can be obtained from Chung, et al., (2006); Perry, et al., (2004); Damian, et al., (2005); Sommerville (2006); and Hoover, et al., (2009).

To implement case methods in software engineering classes, the following steps are suggested:

1. Obtain or design a case. Ensure the case has innovation ideas.
2. If a case study is used, ensure analysis and conclusions are removed.
3. Distribute the case to students and ask them to read it twice, and summarize the main points, ideas and problems related to innovation.
4. During the class meeting, divide the students into groups.
5. Ask each group to state their innovation ideas findings.
6. Ask the groups to explain how the problems can be solved and how the innovation ideas could be potentially marketed.

Some media-rich technical entrepreneurship cases have been developed through the Kern Entrepreneurship Education Network. These cases are available for free online at http://weaverjm.faculty.udmercy.edu/udmkeencases.html. The OnSiteERT involves significant software development cases, which can be provided for a rich discussion around software requirements development and/or user interfaces.

5. Problem-Based Learning

In problem-based learning (PBL), a problem is posed, and knowledge needed to solve the problem is identified. Then a process for learning the required knowledge is initiated. This knowledge could be learned through self-learning, the instructor, or both. Once the knowledge is learned, it will be used to solve the problem. If further knowledge is still missing and is needed, the approach is repeated for that knowledge.

Within PBL, learning is initiated by a problem. Problems are based on complex, real-life conditions. Therefore, the problem should include components that will trigger innovation when solved. Not all information needed to solve the problem is initially provided. Students will work collaboratively to identify, find, and use appropriate innovative ideas to solve the problem. Consequently, PBL will definitely support team-based innovation.
Jeong and Hmelo-Silver (2010) investigated students’ utilization of learning resources in a technologically mediated online learning environment. Undergraduate students were divided into groups, and the groups were involved in an online problem-based learning (PBL) environment, which was equipped with pre-selected video and knowledge resources. Şendag and Odabasi (2009) examined how the online problem based learning (PBL) approach deployed in an online learning environment impacted undergraduate students’ critical thinking skills (CTS) and content knowledge gaining. Their study showed that learning in the online PBL group had a substantial influence on enhancing students’ critical thinking skills.

Problem-based learning has been efficiently applied to our software engineering term projects. The two examples below are taken from projects assigned to our students in software engineering, and software requirements engineering classes respectively.

5.1. Example 1

A company hired your team to engineer a home healthcare software product. The company planned to have this system as the best software in the market. To achieve that, your team will use a suitable process model, set the functional and nonfunctional requirements, design the architectural, user interface, components, and code, write the programs, perform a thorough test, and provide a maintenance plan. The PBL steps are as follows:

1. The following knowledge is identified: home healthcare systems, requirements engineering, architecture design, interface design, component design, code design, programming, testing, and maintenance.
2. Students learn about home healthcare systems and document their understanding.
3. Students learn about requirements engineering (RE) and generate their initial software requirements specification (SRS). After the instructor introduces RE, students modify their SRS.
4. Students learn about architectural design and provide their initial architecture. Later the instructor introduces the architectural design to enable students to come up with their second version of architecture.
5. The same approach above is repeated with the interface design, component design, and code design.
6. Students perform the programming part as they are supposed to have already acquired this knowledge before taking the software engineering class.
7. The instructor introduces software testing.
8. Students test their system using various testing techniques.
9. The instructor explains software maintenance to allow students to complete their maintenance plan.
10. As software engineering is an iterative process, steps could be repeated until the system is finalized.

5.2. Example 2

Online courses are rapidly increasing. Many students are relying on online courses for various reasons. Many educators have reservations regarding online examinations. The face-to-face exams are obviously equipped with strong security measures to catch any cheating. Your team is hired to investigate, write, and analyze the requirements (both functional and nonfunctional) for online examination. Requirements need to be validated and verified. Quality of requirements is a key issue. The PBL is implemented as follows:

1. The needed knowledge includes online exam systems, functional and nonfunctional requirements, requirements validation and verification, and requirements quality.
Strengthening Elements of Teamwork, Innovation, and Creativity in a Software Engineering Program

2. Students read about and understand online exams using the Internet and journal/conference papers.
3. Students study the functional requirements and provide the first version of requirements.
4. The instructor introduces the functional requirements to enable students to modify and improve their requirements.
5. Steps 3 and 4 are repeated for the nonfunctional requirements. As security and privacy requirements are essential for such systems, both the students and instructor will spend more time learning/teaching them.
6. The instructor introduces requirements validation and verification.
7. Students can validate and verify their requirements. As a result, changes are made to the requirements.
8. The instructor introduces requirements metrics.

Students use these metrics to assess the quality of their requirements. Requirements are modified accordingly.

6. Trimming Technique

The goal of the trimming technique, a subset of the well-known TRIZ methods, is to improve a product or introduce a new product based on systematic exploration of the various components, functions, and/or subsystems in a system and to look for ways to generate new ideas (Weaver, 2011a). The Trimming Technique tests and questions our current hypotheses potentially resulting in very interesting and innovative ideas (Verduyn, 2010). The rules for the trimming techniques are given below. In these rules, “function” can refer to a function or method within a program, a requirement within functional and nonfunctional requirements, a component or connector within architecture, a software component, an algorithm, an interface, a test case, or a maintenance procedure. The trimming rules are as follows:

Rule 1: The function is not necessary and should be removed.
Rule 2: The function can be performed by another part/element in the larger system.
Rule 3: The recipient of the function can perform the function itself.
Rule 4: The recipient of the function can be eliminated.
Rule 5: The function can be performed better by a new/improved part providing enhanced performance or other benefits.
Rule 6: A new or niche market can be identified for the trimmed product.
Rule 7: Provide no more of a function than is required.

Trimming is normally applied when a product needs to be differentiated or a new market needs to be created for it, when the product or process is too complex and costly, or when competitors have a patent that needs to be thwarted. The following examples should explain how the trimming technique is used in software engineering. For each example, the applicable rules will be stated.

6.1. Example 1

To apply the trimming technique to the Burglar Alarm System (Figure 1), the following rules may be used:

Rule 1: Voice Synthesizer is not necessary. The Alarm Controller sends the zone numbers in digital form to give the location of the suspected intruder.
Rule 3: If Rule 1 is not used, we can have the Alarm Controller provide either a digital form, a voice form (analog form) or both.
Rule 5: The Movement Sensors will be replaced by Intelligent Sensors that will recognize the residents of the unit and pets. In addition, this new component will detect intruders of different heights.
6.2. Example 2

Given version 1 (Design 1) of the Sales Transaction system in Figure 3, the Trimming Rule is applied as follows:

Rule 1: The link between the Accounting component and the Sales component is not necessary and is removed.

Rule 1: The link between the Accounting component and the Item component is not necessary and is removed.

Rule 5: The Sales component is improved by adding the method “printItemList.”

Figure 4 above provides the improved design (Design 2) for the Sales Transaction system after trimming.

6.3. Example 3

Below is a subset of functional requirements for a Home Healthcare System (Daimi, et al., 2010). Use the Trimming Technique to improve these requirements.

F.1 The system should allow its users to update their personal information.

F.2 The system should allow the primary doctor to update patient information, such as diagnosis, remediation, referrals, health conditions, and prescriptions.

F.3 The system should allow the primary doctor to review the records of a specified patient.

F.4 The system should allow the primary doctor to establish a health plan for a specified patient, which includes the identification of short-term and long-term goals based on a patient’s own specific conditions.

F.5 The system should allow the primary doctor to review and modify the home care plan.

F.6 The system should allow the primary doctor to write prescriptions and forward them to the patient’s pharmacy.

F.7 The system should allow the primary doctor to review the medical prescriptions of a specified patient.

F.8 The system should allow the primary doctor to get health status reports for a specified patient from the caregiver.

F.9 The system should allow doctors to categorize patients into various risk levels so that patients can have different priority-based treatment when an emergency occurs.

F.10 When an emergency event is encountered, the system should automatically generate an alarm to notify the patient’s primary doctor, and, when necessary, the available emergency department (911).

The following rules are applied:

Rule 2: Requirement F3 can be performed by F2. Therefore, F3 is removed.

Rule 2: Requirement F7 can be performed by F2 resulting in F7 being removed.

Rule 5: Requirement F10 can be replaced by an improved requirement that will also call the primary doctor and/or emergency department directly.

7. Bisociation

Bisociation blends the information in multiple domains by discovering a (usually indirect) connection between them. It involves a forced association between the problem at hand and some typically random and seemingly unrelated thing. In other words, bisociation is a link that connects unrelated concepts from two or more domains (Kötter, et al., 2010. Bisociative thinking reconciles the relationship between prior
knowledge and the number of entrepreneurial opportunities recognized (Ko and Butler, 2004). Weaver (2011b) added that bisociation can be a valuable entrepreneurial approach in product creation.

The idea here is to introduce a software engineering artifact, such as a product, a component, a subsystem, or a system, and ask students to write as many ideas as they can come up with within three minutes. An unrelated artifact is then introduced and students are given three minutes to re-write their ideas based on the introduced unrelated concept. Students are then asked to tally their ideas as follows: number of ideas when the original artifact was introduced, number of ideas when the unrelated artifact was introduced, and the number of ideas in the second list but not in the first list. This is then followed by a discussion including questions regarding if students’ thinking has changed when the unrelated artifact was introduced, and if they observe any value in increasing their knowledge in areas outside their current knowledge to improve creativity. According to Weaver (2011b), “Envisioning subtle connections between things that are invisible to others is at the heart of innovation.” The following two examples should exemplify bisociation.

7.1. Example 1

Students are asked to specify the components (ideas) needed for a simple burglar alarm system. Figures 1 and 2 provide some of these components. After three minutes, the instructor will introduce the unrelated stimulus, “sump pump.” Students re-write their ideas within three minutes. Then, the tallying and discussion will start. The idea here is to have students think about extending the alarm system to monitor the sump pump and warn if it fails.

7.2. Example 2

Students are invited to provide ideas on a system that controls the various function of the microwave. At the end of the allowed time, the instructor will introduce the “Internet.” The same process as above is repeated. It should be obvious that the instructor’s goal is to find out if they touch the idea of using the Internet for controlling these functions.

8. Opportunity Recognition

Opportunity recognition comprises searching and encapsulating new ideas that point towards business opportunity. This progression often entails creative thinking resulting in the discovery of new and useful ideas (Katz and Green, 2011). Some of the major issues impacting opportunity recognition are entrepreneurial vigilance, social networks, and personality characteristics (Stritar, et al., 2008). Entrepreneurial vigilance, in its turn, is an essential requirement for the achievement of the opportunity identification triplet: recognition, development, and evaluation (Ardichvili, et al., 2003). Further details regarding opportunity recognition could be found in Arenius and De Clercq (2005); Baron and Ensley (2006); and Klein (2008). Water purification and alarm software systems will be used in the examples below.

8.1. Example 1

The Water Purification (WP) system provides all the functions needed to provide quality water for a resident unit. It will monitor and control the water within the standards set by the health authority. Figure 6 illustrates the functions of the WP system. The Non-Filtered Water Regulator regulates on a daily basis the quantity of water passing through the filter, alerts the user about this quantity, and signals the need to clean or replace filters. The Water Softener Handler controls the monitoring of salt, accepts salt levels
from users, and warns users when salt levels drop below the desired level. The User Notification Manager
is in charge of all user interfaces. Current expectations have evolved to allow the Water Purification
system to include Air Purification. Develop concepts to transform the current Water Purification system
to Water and Air Purification (WAP) system.

8.2. Example 2

The Burglar Alarm System in Figure 2 provides improved burglar monitoring functions. Present
prospects demand the inclusion of gas leak, carbon monoxide, and basement flooding in the monitoring
functions. Develop concepts to transform the Burglar Alarm system to an Integrated Alarm system to
cover all these functions.

9. Innovation Activities Survey

A survey of 12 questions was presented to students at the end of the term. Rating scales of 1-5, where 5
implies “very good” and 1 represents “very poor,” were adopted. Ten students participated in this survey.
The questions of the survey together with the summary of the results are reported in Table 3. The
numbers in each row represent the total number of students that agreed with that rating scale. It is clear
from the table that there was very strong agreement among students regarding the questions 1, 3-7, and
10-11. The strong agreement is calculated by adding the figures for the “5” and “4” rating scales. Here,
at least 80% strongly agreed on each of the above mentioned questions. There was also good agreement
(70%) on questions 2, 8, 9, and 12 using the first two rating scales. For questions 7 and 11, 100% of the
students strongly agreed.

The average opinion for each question was calculated using $\frac{\sum (\text{score} \times \text{weight})}{10}$. The highest average
opinion (4.7) was for question 11. Question 10 was next with an average of 4.4. The lowest average (3.6)
was for question 8. The average opinion for the remaining questions ranged from 4.0 to 4.3. Both
questions 10 and 8 deal with innovative ideas. The mean of their average opinions is 4.0. This is a very
positive indication, as the highest possible average opinion cannot exceed 5.
Strengthening Elements of Teamwork, Innovation, and Creativity in a Software Engineering Program

Table 3. Survey Questions and Results Summary

<table>
<thead>
<tr>
<th>Innovation Activity Experience</th>
<th>Very good</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
<th>Very poor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Connect unrelated concepts from two or more domains</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2. Develop excellent reasoning skills.</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3. Deal with real world examples in the classroom.</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4. Provide the opportunity to students to learn by doing.</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5. Motivate student attention.</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6. Assist in establishing expectations.</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>7. Encourage creative and critical thinking.</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8. Set the foundation for generating innovation ideas.</td>
<td>1</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9. Improve a product based on systematic exploration of the various components.</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10. Looks for ways to generate new ideas.</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11. Improve teamwork discussion and conclusion.</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12. Improve understanding of software engineering.</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

10. Conclusions

Innovation is vital for all nations. If a nation does not innovate, others will, forcing that nation to be dependent on others. In this paper, a number of innovative techniques are introduced. These techniques are applied to various areas of software engineering. In addition to inspiring innovation among our software engineering students, these techniques will enhance students’ motivation and add fun to the software engineering classes applying them.

By starting with painstorming and pinpointing real pain/opportunity in the marketplace, we are ensuring that if we find an innovative solution, it will create real value for customers. This work is at a product development level prior to necessitating detailed financial analysis. Further development of any ideas generated would include financial considerations.

All the problems introduced were solved by teams. Each team was comprised of three students. Although the examples were taken from various software engineering courses, they were unexpected non-traditional exercises and involved considerable ambiguity. All teams failed to come up with the solution during the first attempt. They tried at least twice. They never gave up, and they learned from their failure. Hence, the presented techniques and their accompanying exercises met the following three e-minded attributes: effectively collaborate in a team setting, apply critical and creative thinking to ambiguous problems, and persist through and learn from failure.

It is the aim of these techniques, with the accompanying examples, to encourage faculty to apply these techniques to their classes. The examples provided in this paper could serve as a template for designing further exercises.
Acknowledgement

The author would like to thank Jonathan Weaver, professor of Mechanical Engineering at the University of Detroit Mercy, for his insightful comments that have greatly improved this article.

References


Strengthening Elements of Teamwork, Innovation, and Creativity in a Software Engineering Program


Weaver, J. (2011a). The trimming technique. KEEN Workshop on Shaping Entrepreneurial Engineers. Tempe, AZ.
Weaver, J. (2011b). Bisociation. KEEN Workshop on Shaping Entrepreneurial Engineers. Tempe, AZ.