Encouraging Creative Problem Solving for Aspiring Software Developers

WOUTER GROENEVELD, KU Leuven, Belgium

Software development is a field riddled with complexity, constant change, and above all: technicality. Therefore, much research is devoted to these technical aspects (e.g. cognition), while non-technical skills (e.g. creativity) are given less attention. Our recent work has shown that expert software engineers in industry deem creativity as a crucially important problem solving skill. Yet, we also found that creativity is mentioned in less than 5% of the learning outcomes in computing-related courses across European universities. This denotes a clear gap in skill requirements between education and industry. Our aim is to explore the role of creativity in software engineering, investigating ways to assess and ultimately enhance the creative problem solving skills of computing students in higher education.

CCS Concepts: • **Software and its engineering → Software development techniques**; • **Social and professional topics → Software engineering education**.

Additional Key Words and Phrases: creativity; problem solving; programming; assessment

ACM Reference Format:
Wouter Groeneveld. 2021. Encouraging Creative Problem Solving for Aspiring Software Developers. In *Creativity and Cognition (C&C '21), June 22–23, 2021, Virtual Event, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3450741.3467462

1 INTRODUCTION

Recent computing curricular guidelines slowly but surely started integrating the notion of learning non-technical competencies next to technical knowledge [8]. In computing, technical knowledge usually refers to the ability to use computers to solve a problem, such as knowledge of tools or programming languages. The word "technical" comes from the Greek "Techné", which means trade. Examples of technical computing-related courses are compilers, distributed networks, operating system concepts, and software engineering. As critical as trade-specific skills that can tame a computer are for software developers, mastering only technicality is not enough [5]. So, a second group of skills we call "non-technical" are also required to succeed as a developer. However, next to the perhaps obvious examples such as teamwork and communication skills, we wonder:

What exactly are these non-technical requirements for modern software engineers? (RQ1)

To answer this question, we conducted a Delphi study, in which 36 experts from 11 different countries world-wide were invited [5]. To emphasize the relation with industry, this paper was co-authored by a senior developer (second author). A list of non-technical skills that software engineering (SE) experts think is relevant to succeed as a developer was synthesized. The list contains 55 ranked skills, classified in four major areas: *communicative skills* (empathy, actively listening, etc.), *collaborative skills* (sharing responsibility, learning from each other, etc.), *problem solving skills* (verifying assumptions, solution-oriented thinking, etc.), and *personal skills* (curiosity, being open to ideas, etc.). Creativity was found to be one of the highest ranked problem solving skill.

In the ACM/IEEE Computer Science Curricula of 2020, the terms “creativity” and “creative” are collectively only mentioned 14 times, while for instance “communication” is mentioned 84 times in the document [8]. This signifies the presence of a gap between what industry experts think is important and what is currently being taught in higher education. Therefore, we ask:

What are the differences between the requirements of RQ1 and SE education? (RQ2)

First, in 2019, we conducted a systematic literature review (SLR) to analyze teacher success stories of non-technical skills [7]. The SLR shows self-reflection, conflict resolution, communication and teamwork as the four most taught non-technical skills. We found little papers that utilize internships and capstone projects as a full-fledged teaching aspect to facilitate the learning of multiple skills, including creativity. Next, in 2020, we manually curated 278 non-technical syllabi from 110 universities in 30 European countries by scraping the learning outcomes of course information websites [4]. The most frequently identified skills are teamwork, ethics, written/oral communication, and presentation skills, while the development of one’s own values, motivating others, creativity, and empathy feature least frequently. Again, creativity was somehow neglected, compared to other skills.

A thorough analysis of all the data collected so far showed that creativity is one of the most sought after skills in industry, and yet one of the least frequently present in both our SLR and learning outcomes. A synthesis of the gap analysis that answers RQ2 has been submitted for review. We hope that our efforts on the subject not only sheds more light on to the curricular shortcomings within computing higher education, but also contributes to a solution of this teaching gap. Our last research question is:

How can we improve upon education to reduce the gap of RQ2? (RQ3)

Since RQ1 and RQ2 have already been answered, of which several works have been peer-reviewed and published, this paper focuses on zooming in on RQ3, describing past, current, and future research plans. In the coming years, we will focus on exploring, analyzing, and assessing creativity in software engineering education.

2 EXPLORING CREATIVITY IN SOFTWARE ENGINEERING

The first step towards answering RQ3 has been made by exploring creativity in the world of software development through the organization of four focus groups. We invited 33 participants to creatively brainstorm about creativity [6]. The paper is accepted and will be presented at the ACM/IEEE International Conference on Software Engineering (ICSE) in May 2021. To emphasize the the interdisciplinarity of the research, we solicited the help of someone involved in Architecture and Design (second author).

The 399 minutes of transcripts, coded into 39 sub-themes, were grouped into seven categories: *technical knowledge*, *communication*, *constraints*, *critical thinking*, *curiosity*, *creative state of mind*, and *creative techniques*. This study identifies the added value of creativity, which creative techniques are used, how creativity can be recognized, the reasons for being creative, and what environment is needed to facilitate creative work. A mind map of the seven domains and each identified sub-theme is visible in Figure 1. The mind map is the result of several focus group data processing steps we went through: (1) audio transcribing, (2) open coding, (3) axial coding, and (4) merging results. More details about the focus group methodology are available in [6].

At the heart of this mind map lie the keywords *creative problem solving*. This is purposefully not entitled *creativity*. We are interested in the practicality of creativity while encountering programming problems, and how to take the hurdle in order to solve the problem at hand. We are less interested in the psychological dimensions of creativity, such as a possible link to the Big Five personalities, which has been proven before to be related [11].

Fig. 1. A mind map of the seven identified domains of creative problem solving in SE.

Furthermore, many published works within the computing education community have a surprisingly narrow vision of the word *creativity*. It will mostly be equaled to *divergent thinking*, and linked to the Torrance Tests of Creative Thinking [13]. We came to that conclusion after conducting a second systematic literature review to support research specifically on creativity in computing education by summarizing relevant theories, instruments, and other prior work in this area. The paper is currently under review.

While we agree that divergent thinking is an integral part of creative problem solving (in the mind map: part of the *Creative Techniques* domain), we are convinced that in order to confidently solve problems, much more is required. Therefore, we opted for "creative problem solving" (CPS) instead of utilizing the higher level term "creativity"[1].

The seven identified domains of CPS are inherently linked to each other. Without curiosity, there would be little intrinsic incentive to learn more technical knowledge through continuous learning. Without critical thinking, one would not be able to discern quality from rubbish while constantly seeking out different inputs. Without the right creative state of mind, brainstorming sessions would yield little new ideas.

After a first exploration of the role of creativity in software engineering, we now turn back to the skill gap in higher education. How can we integrate the findings from the focus group study into the computing curriculum (RQ3)? Before introducing any changes, we are currently developing a CPS survey that can self-assess the skill level of students on the identified domains.

[1]We refrained form using "creative coding" as it usually refers to employing code to express oneself, not to solve a programming problem at hand.

3 ASSESSMENT OF CREATIVE PROBLEM SOLVING

As part of a gentle introduction into academic research, we let a group of second-year students evaluate the creativity of first-year students' Software Design in Java programming projects, based on Amabile's Consensual Assessment Technique (CAT) [1]. We then introduced students to PMD[2], a static code analysis tool. We were interested in seeing whether the more creative projects are also the technically better ones. PMD reports on several code quality issues, enabling us to correlate creativity with clean code. We concluded that there is indeed a (moderate) correlation: the more creative, the more code quality issues. This signifies the importance of also teaching about *clean code* instead of simply introducing CPS in programming courses. The paper is currently under review.

Assessing the creativity of a programming project was done by giving a score on the tangible end product: the output of students' CPS efforts. However, our aim is to inspect how to teach, inspire, and improve all levels of problem solving skills. It could very well be that students showed great great levels of curiosity and employed many creative techniques, even if the end product ended up scoring lower compared to works of other students. It was clear to us that relying on something like CAT is insufficient.

Instead, we are currently developing a tool students can use to self-assess their creativity based on the seven domains identified in [6]. As part of a pilot study, we have collected results from 140 first-year engineering students. The questions are heavily geared towards practicality: students are asked to answer the questions in context of their CS1 programming assignment. For each of the seven domains, we developed eight questions, based on several existing published and validated surveys, such as the Critical Thinking Disposition Scale [10] and the Curiosity Index [3].

We are currently investigating the reliability and validity of the developed scale. Our aim is to reduce the question set without sacrificing the internal item-total relationships between each factor. A preliminary factor analysis hinted at the presence of three major factors. Thus, we could further synthesize the seven domains of CPS into for example the following three concepts:

1. **Ability**: *Knowledge*, *Critical Thinking*, and *Constraints*.
2. **Mindset**: *Curiosity* and *Creative State of Mind*.
3. **Doing**: *Communication* and *Creative Techniques*.

After the first results of the pilot study have been published, we will proceed to re-iterate on the survey. Future work will involve using the validated survey to measure the effectiveness of our efforts to improve upon computing education by amplifying CPS for aspiring software developers.

4 FUTURE WORK

Before being able to introduce changes, we first plan to develop a theoretical framework that explains, supports, and supplements CPS in the world of software development. At this moment, rough outlines of this theory are being set. A planned trajectory of work toward thesis completion is summarized in the timeline of Figure 2. The thesis work started in September 2018 and is planned to be defended in June 2023. The research part of the academic appointment takes up 50%, together with half-time teaching activities. The timeline contains four different tracks:

1. Survey work (turquoise). Rework the survey if necessary (subdivided into two blocks, flag B) based on our findings, or gather more data for the next step of the survey validation.
2. Research (purple). Setting up the theoretical framework and partaking in working groups on global competencies in computing education.

[2] See https://pmd.github.io/.

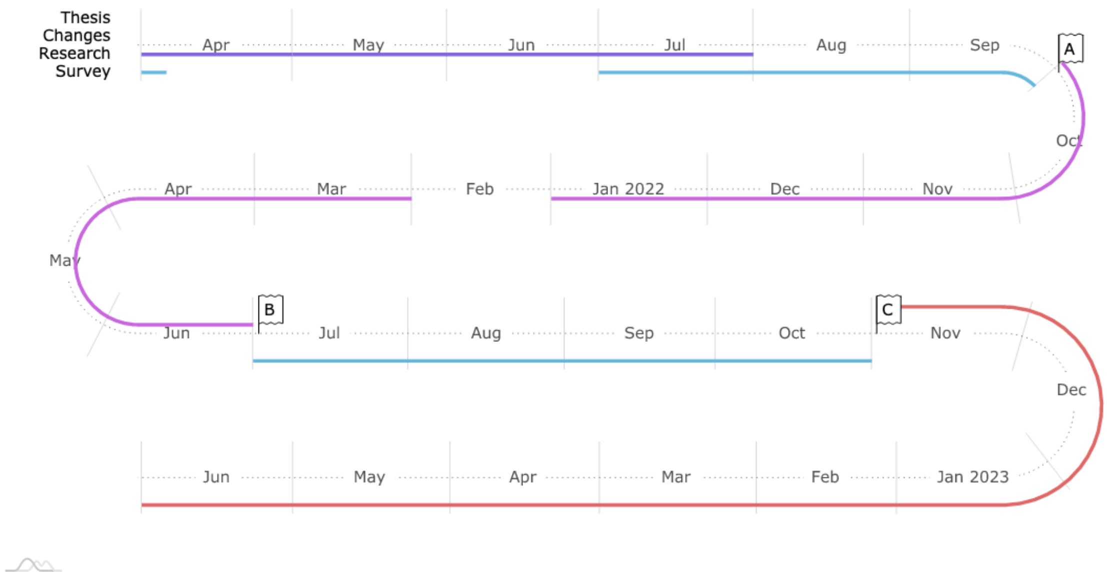

Fig. 2. A timeline of future planned work.

3. Introducing changes (pink). Putting the test to the test by using it to measure pre and post intervention. We plan to intervene twice (flag A), based on iterative feedback, possibly on other campuses.
4. Thesis preparation (orange, flag C). Prepare final intervention recommendations for creative problem solving in SE and start writing the thesis itself.

We are very much aware of the multidisciplinarity of creativity research. We hope the ACM Creativity & Cognition Conference will provide the needed fertile ground on which ideas on creativity in computing education can further flourish. The aim for this paper is to receive feedback and discuss possibilities we might otherwise have missed because of our technical background. Hopefully, possible interdisciplinary collaborations will emerge. We are convinced that the only way for us to move forward in this research is to work across the Engineering Technology (our faculty) border.

4.1 Amplifying creative problem solving in computing education

As for the possible changes in education itself, we are currently investigating the possibilities to integrate them into the teaching programme in consultation with the coordinating responsible people at our local Campus. The following ideas are currently being taken into consideration:

- Explicitly introducing students to the seven CPS domains within an existing course. Other studies in cognitive and creative thinking show that bringing these models into the classroom can already introduce mind shifts [9].
- Employing Bruce A. Tate's Seven Languages in Seven Weeks [12] concept to touch upon all the problem solving domains, whether in a separate experimental course or in a the form of multiple successive workshops.

- Exploring Jessica Kerr's Symmathecist "collective problem solving" mutual learning ideas[3] in code labs through real-life cases by inviting experts from industry.
- Doing code kata's via mob programming to re-implement the same problem again and again, viewing the same problem from many different angles, using different techniques, and combining several ideas.
- Explicitly teaching multiple design-based depth-first and breadth-first approaches to a programming problem, and the mediating role uncertainty has in this, as presented in [2].
- ...

5 CONCLUSION

This paper summarizes the thesis work that consists of three overarching research questions: (1) what are non-technical requirements for modern software engineers, (2) how big is the requirement gap between industry and software engineering education, and finally (3) how can we improve upon education to reduce that gap. RQ1 and RQ2 both pointed to creativity and the lack thereof, causing us to focus on creative problem solving in order to answer RQ3. We hope the Creativity & Cognition Conference will broaden our perspective of creativity and spark ideas that can be transferred to the world of computing education.

REFERENCES

[1] Teresa M Amabile. 1988. A model of creativity and innovation in organizations. *Research in organizational behavior* 10, 1 (1988), 123–167.

[2] Linden J Ball, Balder Onarheim, and Bo T Christensen. 2010. Design requirements, epistemic uncertainty and solution development strategies in software design. *Design Studies* 31, 6 (2010), 567–589.

[3] Keston H Fulcher. 2004. *Towards measuring lifelong learning: The curiosity index.* Ph.D. Dissertation. ProQuest Information & Learning.

[4] Wouter Groeneveld, Brett A Becker, and Joost Vennekens. 2020. Soft Skills: What do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 287–293.

[5] Wouter Groeneveld, Hans Jacobs, Joost Vennekens, and Kris Aerts. 2020. Non-cognitive abilities of exceptional software engineers: a Delphi study. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1096–1102.

[6] Wouter Groeneveld, Laurens Luyten, Joost Vennekens, and Kris Aerts. 2021. Exploring the Role of Creativity in Software Engineering. In *2021 IEEE/ACM 43nd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE/ACM.

[7] Wouter Groeneveld, Joost Vennekens, and Kris Aerts. 2019. Software engineering education beyond the technical: A systematic literature review. In *Proceedings of the 47th Annual SEFI Conference*. 1607–1622.

[8] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2020. *Computer Science Curricula 2020: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Technical Report. New York, NY, USA.

[9] Tricia J Ngoon. 2019. Overcoming Satisficing: Scaffolds for Amplifying Creativity. In *Proceedings of the 2019 on Creativity and Cognition*. 670–674.

[10] Edward M Sosu. 2013. The development and psychometric validation of a Critical Thinking Disposition Scale. *Thinking skills and creativity* 9 (2013), 107–119.

[11] Sun Young Sung and Jin Nam Choi. 2009. Do big five personality factors affect individual creativity? The moderating role of extrinsic motivation. *Social Behavior and Personality: an international journal* 37, 7 (2009), 941–956.

[12] Bruce Tate. 2010. *Seven languages in seven weeks: a pragmatic guide to learning programming languages*. Pragmatic Bookshelf.

[13] E Paul Torrance. 1972. Predictive validity of the torrance tests of creative thinking. *The Journal of creative behavior* (1972).

[3]See https://jessitron.com/2018/10/25/symmathecist-n/. These ideas will be merged in the theoretical framework that will be developed as an aid to introduce the changes.